E.T.S. de Ingeniería Industrial, Informática
y de Telecomunicación

Diseño e implementación de la comunicación entre un sistema IoT de control de calidad del aire y una aplicación .NET Core 5

Grado en Ingeniería Informática

Trabajo Fin de Grado

Mónica Campos Romeo
Adrián Catón Oteiza
Francisco Javier Falcone Lanas

Pamplona, 11 de junio de 2021

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Índice

Resumen

Este proyecto es en colaboración con la empresa i3Code Solutions, quien ha recibido de varios de sus clientes la consulta de cómo implementar soluciones de IoT y si éstas pudieran integrarse con las aplicaciones que ya disponen. Este es un tema en el que i3Code no ha indagado mucho y por ello le parece un tema muy interesante. Además, les gusta la idea de poder incrementar sus soluciones con integraciones IoT y poder comercializar con ellas.

Por todo ello, se ha planteado como objetivo de este proyecto crear esa implementación de una solución IoT con una aplicación web como las que desarrolla i3Code. Se ha tenido claro que el dispositivo IoT a utilizar iba a ser un dispositivo que detectara la calidad de aire, porque debido a la situación actual de pandemia de Covid-19 que se está viviendo, se ha detectado lo importante que es conocer la calidad de aire del interior de los lugares tanto públicos como privados y más en concreto de oficinas, viviendas, comercios, etc. Por lo que i3Code ha decidido apostar no solo por implementar la solución pedida por los clientes sino también por tratar de crear una solución comercializable.

Por tanto, el presente proyecto tiene como objetivo integrar y programar dispositivos de IoT para detectar la calidad del aire en interiores, así como diseñar e implementar la comunicación entre estos dispositivos y una aplicación .NET Core 5 estudiando también las tecnologías y protocolos que pueden ser utilizadas en el desarrollo de dicha comunicación.

Abstract

This project is in collaboration with the company i3Code Solutions, which has received from several of its customers a consultation on how to implement IoT solutions and whether they could integrate with the applications they already have. This is a topic that i3Code has not researched very much and therefore it is a topic very interesting for them. In addition, they like the idea of being able to increase their solutions with IoT integrations and be able to market with them.

As an object for this project, it has been proposed to create an implementation of an IoT solution with a web application such as those developed by i3Code. It has been clear that the IoT device to use was going to be a device that would detect air quality, due to the current situation of the Covid-19 pandemic that is happening, it has been known how important it is to know the air quality inside both public and private places and more specifically offices, houses, businesses, etc. i3Code has decided to bet on implementing the solution ordered by customers and trying to create a marketable solution.

Therefore, this project aims to integrate and program IoT devices to detect indoor air quality, as well as to design and implement communication between these devices and a .NET Core 5 application by also studying the technologies and protocols that can be used in the development of such communication.

Lista de palabras clave

Aplicación Web: herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o una Intranet mediante un navegador web.

Co2 (dióxido de carbono): Gas inoloro e incoloro que se producido al respirar.

Módulo de Arduino: placa que se puede conectar a una tarjeta Arduino para añadirle más funcionalidades.

PPM (partes por millón): unidad de medida de la concentración que mide la cantidad de unidades de soluto por cada millón de unidades de solución.

Protocolo: conjunto formal de estándares y normas que rigen el formato y el control de la interacción entre diferentes dispositivos.

Publish/Subscribe: protocolo de comunicación que permite que cualquier número de suscriptores reciban información de forma asíncrona y anónima de los publicadores. Utiliza colas de mensajes y se asocia con middlewares basados en el paso de mensajes.

Request/Response: método básico utilizado para la comunicación de dos máquinas en el que un integrante de esa comunicación solicita una información y el otro se la entrega.

Sistema IoT: sistema de internet de las cosas, permite conectar elementos físicos cotidianos a Internet.

Wifi: estándar 802.11 que permite la conexión inalámbrica a Internet de diferentes dispositivos.

Acrónimos y abreviaturas

Acrónimo	Descripción
ACI	Azure Container Instances
AKS	Azure Kubernetes Service
API	Application Programming Interfaces
ASP	Active Server Pages
AWS	Amazon Web Services
CD	Continuous Delivery
CI	Continuous Integration
CoAP	The Constrained Application Protocol
CRUD	Create, Read, Update and Delete
CSS	Cascading Style Sheets
DNS	Domain Name System
DTLS	Datagram Transport Layer Security
HPC	High Performance Computing
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IoT	Internet of Things
JS	JavaScript
JSON	JavaScript Object Notation
M2M	Machine to machine
MQTT	Message Queue Telemetry Transport
MVC	Modelo-Vista-Controlador
OCI	Open Container Initiative
ORM	Object-Relational Mapper
PaaS	Platform As A Services
QoS	Quality of Service
RAML	RESTful API Modelling Language
REST	Representation State Transfer
SAP	Systems, Applications, Products in Data Processing
SDK	Software Development Kit
SOA	Service Orientated Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus

W3C	World Wide Web Consortium
WADL	Web Application Description Language
WDSL	Web Services Description Language
WSN	Wireless Sensor Networks
XML	Extensible Markup Language
XMPP	The Extensible Messaging and Presence Protocol

Tabla 1: Tabla de acrónimos y sus descripciones

1. Introducción

El mundo de las tecnologías IoT se ha vuelto algo tan común en nuestro día a día que es normal que las empresas decidan implementar estos dispositivos. Cuentan con infinidad de posibles implementaciones y son capaces de hacer infinidad de tareas. También es algo lógico que estas empresas quieran incorporar estos dispositivos con las soluciones de software que ya dispongan y en las que ya hayan invertido dinero en su fabricación y tiempo tanto en la espera a que se completen como en entender y comprender su funcionamiento.

Esto se ha podido comprobar con la empresa i3Code, la cual ha percibido en sus clientes la incertidumbre de si se pudieran añadir estos dispositivos IoT en las soluciones que ya han implementado con ellos. Por esto mismo i3Code ha decidido apostar por formarse y conocer cómo se realizan este tipo de implementaciones y tratar de conseguir un posible producto comercializable con integración en sus aplicaciones web.

Por otro lado, la situación actual de la pandemia de Covid-19 ha hecho que se tenga mucho más en cuenta la calidad del aire y la ventilación de los espacios. Y es que un alto nivel de Co2 en una habitación puede indicar que ésta se encuentra mal ventilada y por tanto es una zona más propensa a contagios de enfermedades que tienen como medio de transmisión el aire. Entre estas enfermedades se encuentra la Covid-19. Esto es algo que también tiene muy en cuenta la empresa i3Code y después de todo este tiempo querían poder ayudar en la situación actual de la pandemia.

2. Justificación y objetivos

Juntando las dos situaciones comentadas en la Introducción, desde i3Code se tuvo muy claro cuáles eran los pasos a seguir y lo convirtieron en el proyecto actual. La solución para apostar por la integración de dispositivos IoT en sus soluciones y ayudar en la pandemia, era la de crear un dispositivo IoT que controlara la calidad del aire e integrar esto con una solución del tipo de soluciones que realiza la empresa i3Code con sus clientes.

Con este proyecto se pretenden solucionar las dos ideas de i3Code, al mismo tiempo que tratar de crear una solución comercializable por parte de la empresa de forma individual. Para ello, se va a integrar y programar el dispositivo IoT de detección de calidad de aire en interiores, al mismo tiempo que diseñar e implementar la comunicación de este con una aplicación .NET Core 5, como desarrolla i3Code para sus clientes. Esta aplicación web ha de ser funcional independientemente del resto de soluciones de i3Code para poder comercializarla de forma individual.

Para todo esto, se tienen que estudiar las tecnologías y protocolos que pueden ser utilizadas en el desarrollo de la comunicación, así como qué hardware es el adecuado para crear un sistema de calidad de aire eficiente. Una vez estudiado se procederá a desarrollar todo el sistema y a obtener una solución final funcional.

3. Estado del arte

A la hora de realizar la solución hay que plantearse cuál de todas las opciones del mercado se va a utilizar para construir el dispositivo IoT, realizar la comunicación entre el dispositivo y la red, implementar la solución software, etc.

3.1 ¿Qué dispositivo IoT utilizar?

Para esto, lo primero es encontrar un dispositivo acorde con la solución. La empresa i3Code, con la que se realiza este proyecto, dispone de una placa Arduino Uno R3 por lo que se va a utilizar esta misma, es una de las placas más sencillas de Arduino, pero dispone de toda la tecnología que se necesita en este proyecto.

Ilustración 1: Placa Arduino UNO R3

Para la detección de la calidad del aire, los sensores de gases más comunes son los sensores MQ2 y MQ135, cada uno de ellos está especializado en la detección de diferentes gases. El sensor MQ2 está especializado en la detección de GLP, propano, metano, alcohol, hidrógeno y humo, siendo más sensible al GLP y propano. En cambio, el sensor MQ135 se utiliza en equipos de control de calidad del aire para edificios y oficinas y es adecuado para la detección de NH3, NOx, alcohol, benceno, humo, CO2, etc. [1] Por esto mismo, el sensor que se debe utilizar es el MQ135.

Ilustración 2: Sensor CO2 MQ135

Para enviar los datos del sensor a la red es necesario disponer de conexión a la misma, en este caso se va a utilizar WIFI. La placa Arduino UNO R3 no dispone de wifi, por lo que es necesario un módulo WIFI adicional. Los módulos WIFI más usados son los módulos ESP8266, sus diferentes tipos van desde el ESP-01 al ESP-14. Las diferencias son el número de pines y si disponen o no de antena, pero todas integran el mismo chip. Los módulos más usados son el ESP-01 y el ESP-12, la principal diferencia es el número de pines a los que se puede acceder, así

como que existen módulos que traen el ESP-12 y conexión directa a USB, esto es un incentivo bastante importante, pero los pines del módulo ESP-01 son suficientes para este proyecto. En caso de necesitar más pines se estudiaría la posibilidad de utilizar un módulo ESP-12.

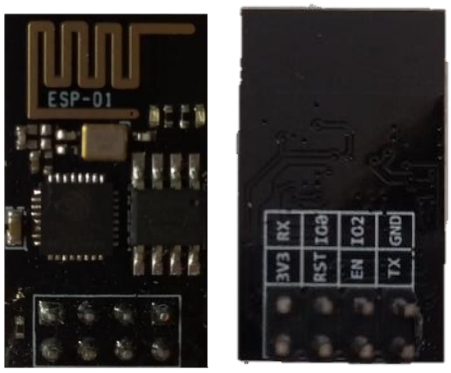

Ilustración 3: Modulo Wifi ESP8266 ESP-01

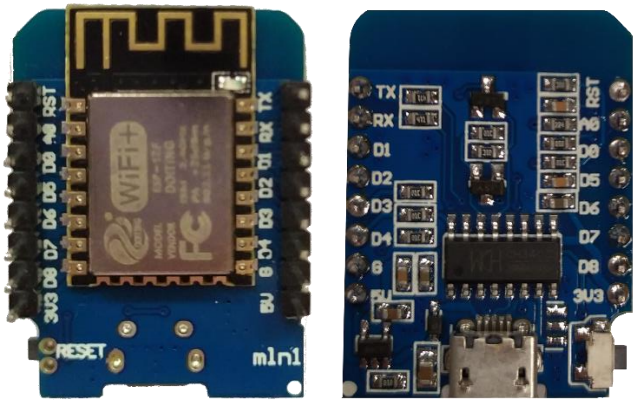

Ilustración 4: Modulo Wifi ESP8266 ESP-12

En conclusión, los dispositivos que se van a utilizar en el proyecto son un Arduino, un sensor Co2 y un módulo Wifi. El Arduino simplemente hace la función de adaptador para que el módulo wifi se conecte con el ordenador y pueda ser programado y enviar datos. Por lo tanto, el Arduino (ver Ilustración 1) se utilizará con el microprocesador desconectado o con un cable desde el pin GND al pin RST. Esto se hace para utilizar los recursos de los que dispone la empresa i3Code Solutions, pero cualquier otro tipo de conector serviría de la misma forma.

3.2 ¿Qué protocolo utilizar para la comunicación entre el dispositivo y la red?

Los protocolos más utilizados en el ámbito de IoT para la transmisión de mensajes entre dispositivos son: **MQTT**, **CoAP**, **XMPP** y **HTTP**.

	MQTT	CoAP	XMPP	HTTP
Modo	Publish/Subscribe	Cliente/Servidor	Cliente/Servidor	Cliente/Servidor
Transporte	TCP o UDP	UDP	TCP o HTTP	TCP
Seguridad	SSL/TLS	DTLD	SSL/TLS	SSL/TLS

Tabla 2: Comparativa entre protocolos

MQTT fue lanzado por IBM con el objetivo de crear una comunicación de M2M. Se trata de un protocolo de capa de aplicación asíncrono de Publish/Subscribe y se transmite sobre el protocolo de transporte TCP. Tiene como objetivo conectar dispositivos y redes integrados con aplicaciones y middleware utilizando el protocolo Publish/Subscribe para proveer de flexibilidad en las transiciones y una implementación simple. El método que utiliza es Topic-based.

Representa un protocolo de mensajería ideal para las comunicaciones IoT y M2M y es capaz de proveer un enrutado para dispositivos pequeños, baratos, de bajo consumo y poca memoria en redes vulnerables y con pequeño ancho de banda. En el protocolo MQTT, existe un bróker o servidor que contiene una serie de temas y cada cliente puede ser un Publisher que manda información al broker sobre un tema específico y/o un subscriptor que recibe automáticamente un mensaje cada vez que se actualiza o publica información en un tema al que está subscrito.

MQTT es muy liviano y, por lo tanto, adecuado para escenarios M2M, WSN y, en última instancia, de IoT, donde los nodos de sensores y actores se comunican con aplicaciones a través del agente de mensajes MQTT. El protocolo fue diseñado específicamente para aplicaciones de telemetría remota, con tres objetivos específicos de diseño:

1. Debe ofrecer un modo de entrega "solo una vez" asegurado, para permitir que un mensaje se transfiera de forma fiable desde un sensor remoto hasta una aplicación back-end.
2. El protocolo debe ser lo más ligero posible a través del cable (u otro medio de comunicación). La mayor parte de la telemetría remota se realiza en redes de bajo ancho de banda, redes de alto coste, por lo que es muy deseable minimizar la sobrecarga de cada mensaje.
3. El protocolo debe ser muy fácil de implementar en dispositivos integrados, como sensores y salidas.

En conclusión, MQTT es un protocolo de cableado centrado en el mensaje y diseñado para la comunicación M2M. Permite la transferencia de datos de tipo telemétrico en forma de mensajes desde los dispositivos, a lo largo de redes de alta latencia o redes restringidas, a un servidor o a un pequeño intermediario de mensajes. Los dispositivos pueden ser, desde sensores y actuadores, hasta teléfonos móviles, sistemas integrados en vehículos y ordenadores completos. Soporta el protocolo de comunicación Publish/Subscribe y es muy simple.

	Bit	0	1	2	3	4	5	6	7
Área fija	Byte 1	Tipo de paquete MQTT				Flags del paquete MQTT			
	Byte 2	Tamaño restante del paquete (sin contar el tamaño de la parte fija)							
Área Variable	Byte 3	Identificador de paquete							
	Byte 4								
	Byte 5	Tamaño de propiedades							
	Byte 5 ⋮ Byte n	Propiedades (los id)							
	Byte n+1 ⋮ Byte m	Payload (mensaje)							
	Byte m+1	Resultado del envío del mensaje							

Tabla 3: Diseño del paquete MQTT

CoAP es un protocolo de capa de aplicación síncrono de petición/respuesta. Tiene como objetivo permitir que pequeños dispositivos con poca potencia, poca capacidad de computación o cálculo y poca comunicación, utilicen interacciones RESTful. Ha permitido dotar a los dispositivos con recursos limitados de servicio de web. Es un protocolo binario que se ejecuta sobre el protocolo de transporte UDP. Una subcapa de mensajería añade una fina capa de control que proporciona la detección de duplicados y, opcionalmente, la entrega fiable de mensajes basada en un sencillo mecanismo de parada y espera para las retransmisiones. Se trata del protocolo request/response que utiliza tanto respuestas síncronas como asíncronas. La razón por la que se ha diseñado sobre UDP es eliminar las cabeceras TCP y reducir el ancho de banda necesario para la comunicación. Aunque fue creado para IoT y comunicaciones M2M, no incluye ninguna funcionalidad de seguridad integrada.

	Bit	0	1	2	3	4	5	6	7
Cabecera	Byte 1	Versión		Tipo		Longitud Token			
	Byte 2	Clase de código				Detalles del código			
	Byte 3	Id del mensaje							
	Byte 4								
Área Variable (no aparece siempre)	Byte 5 ⋮ Byte n	Token (relaciona la petición con la respuesta, ocupa máximo 8 bytes)							
	Byte 5 ⋮ Byte n	Opciones (si aparece, la parte fija ocupa entre 1 y 5 bytes)							
	Byte n + 1	Marcador de payload (marca el comienzo de este si existe)							
	Byte n+1 ⋮ Byte m	Payload (mensaje)							

Tabla 4: Diseño del paquete CoAP

XMPP fue diseñado para chatear e intercambiar mensajes. Fue estandarizado por la IETF hace una década, por lo que es un protocolo bien probado que se ha utilizado ampliamente en todo Internet. Google dejo de apoyarlo por la falta de apoyo mundial. Utiliza el protocolo

cliente/servidor. XMPP es seguro y permite la adición de nuevas aplicaciones sobre los protocolos principales. Al tratarse de un protocolo diseñado para el envío de mensajes no se va a tener en cuenta su diseño.

HTTP fue diseñado entre 1989 y 1991. Ha ido evolucionando hasta llegar a la versión HTTP/2. Se trata de un protocolo de capa de aplicación síncrono de cliente/servidor que implementa request/response siendo la base de cualquier intercambio de datos en la Web. La comunicación es iniciada por el cliente que realiza la petición y es el servidor el que le da la respuesta. HTTP se transmite sobre el protocolo de transporte TCP. En la mayoría de los casos el cliente es un navegador web, pero puede ser también un dispositivo IoT. Algunos de los beneficios que aporta HTTP es su sencillez, ya que está desarrollado para ser fácilmente interpretado por personas y su extensibilidad, debido a sus cabeceras y sus conexiones, ya que se trata de un protocolo que no necesita conexión continua, solamente necesita un protocolo de transporte fiable como es TCP. Uno de sus inconvenientes puede ser el hecho de que no guarde el estado, por lo que necesita de cookies o variables de sesión para poder guardar datos entre dos peticiones en la misma sesión.

<table>
<tr><th></th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td rowspan="4">Cabecera</td><td>Byte 1
⋮
Byte 7</td><td colspan="8">Content-Length (tamaño de la trama)</td></tr>
<tr><td>Byte 8</td><td colspan="8">Tipo de trama (define como se va a interpretar)</td></tr>
<tr><td>Byte 9</td><td colspan="8">Flags del tipo de trama</td></tr>
<tr><td>Byte 10
⋮
Byte 18</td><td>R</td><td colspan="7">Content-Length (tamaño de la trama)</td></tr>
<tr><td></td><td>Byte 19
⋮
Byte n</td><td colspan="8">Payload (mensaje)</td></tr>
</table>

Tabla 5: Diseño del paquete HTTP

WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP, por lo que trabaja a nivel de transporte. Está muy relacionado con HTTP y efectivamente lo utiliza para establecer la conexión. Esta conexión se mantiene activa para el envío de mensajes de forma bidireccional. WebSocket permite recibir los datos directamente al navegador web, esos datos mantienen el orden de entrega de los mensajes.

Para el caso que ocupa este proyecto es importante la fiabilidad del envío de paquetes, algo que CoAP no asegura ya que trabaja sobre el protocolo no seguro UDP, y también es importante que el protocolo utilizado sea adecuado para los dispositivos IoT como lo son MQTT y HTTP. En el caso de WebSocket es una tecnología más dedicada a la comunicación a través de web, por lo que ahora, para decidir el mejor protocolo para este proyecto, se va a evaluar el rendimiento de MQTT y HTTP.

En cuanto al rendimiento, se va a analizar lo siguiente [2]:

3.2.1 Sencillez y ligereza

La implementación de MQTT es muy sencilla y liviana ya que basta con conocer los tipos de mensajes que se pueden enviar para el protocolo Publish/Subscribe. En cambio, HTTP

requiere de más conocimientos, aunque esto puede ser un beneficio para aquellas comunicaciones que requieran de más control.

En cuanto al tamaño de los paquetes, tanto MQTT como HTTP trabajan sobre paquetes TCP, por lo que cuentan con un tamaño base de 20 bytes que son las cabeceras TCP. Pero teniendo en cuenta solo la extensión del paquete MQTT o el paquete HTTP, el paquete MQTT tiene una extensión entre 2 y 12 bytes sin contar la extensión del mensaje a enviar. En cambio, HTTP tiene una extensión fija de 17 bytes sin contar la extensión del mensaje a enviar.

En cuanto al tipo de mensaje que es capaz de enviar cada protocolo, MQTT puede transportar los mensajes en crudo, es decir, en binario, en cambio HTTP debe enviarlos codificados en base64, por lo que el mensaje en el protocolo HTTP es algo mayor ya que aumenta el tamaño del mensaje alrededor de un 33%.

Otra diferencia entre ambos protocolos es que MQTT se trata de una conexión TCP continua, es decir, siempre abierta, por lo que el dispositivo IoT debe estar siempre alimentado a cambio de recibir las modificaciones al instante. Ya que cualquier cambio en la configuración o los mismos datos, se reciben de forma instantánea en el broker MQTT quién los envía a los subscriptores. En cambio, HTTP se trata de una conexión TCP que se abre cuando se van a enviar los datos, y para conocer los cambios de configuración o los mismos datos, el receptor debe estar permanentemente consultándolos. Esto hace que el dispositivo IoT no necesite estar conectado a la red ya que con la alimentación de una pila o batería sería suficiente, porque consume muchos menos recursos que la conexión MQTT.

3.2.2 Velocidad

Según la información encontrada [3] MQTT es de media 20 veces más rápido y requiere 50 veces menos tráfico que HTTP a la hora de publicar datos consistentes y valiosos en el tiempo. La parte de servicio MQTT requiere solo un 10% menos de tráfico que HTTP, siendo la ventaja del servicio de MQTT de la red cableada frente a la inalámbrica insignificante. En cuanto a la eficiencia energética, MQTT, incluso a corto plazo, es un 22% más eficiente energéticamente y un 15% más rápido y vuelve a no haber diferencia entre las redes cableadas e inalámbricas. Esto último es un punto positivo ya que en el caso que ocupa este proyecto se va a utilizar una red inalámbrica a través de un módulo Wifi.

3.2.3 Fiabilidad

En cuanto a la calidad de servicio (QoS en inglés) MQTT ofrece tres diferentes niveles: QoS 0, QoS 1 y QoS 2.

- QoS 0 realiza como mucho una entrega, se trata de una entrega “best effort” o mejor esfuerzo, ya que realiza únicamente una entrega sin tener en cuenta si se recibe o no, por lo que no cuenta con ninguna garantía frente a fallos.
- QoS 1 realiza al menos una entrega ya que se asegura mediante ACKs (mensajes de confirmación) que ha llegado un paquete, aunque puede haber duplicados.
- QoS 2 realiza solo una entrega ya que garantiza que cada mensaje se reciba una única vez, sin duplicados. Esto puede ser interesante después de realizar alguna actualización, ya que es importante que ese mensaje llegue al destinatario.

En el caso de HTTP no implementa la calidad de servicio por sí mismo.

MQTT cuanta con la función de “última voluntad” (Last will & Testament en inglés) que, en el caso de una desconexión repentina de un cliente, los clientes suscritos al tema que

publicaba ese cliente reciben una notificación por parte del MQTT broker. Por otro lado, cuenta con la "retención de mensajes" que permite a un cliente recién suscrito a un tema conseguir una actualización de estado inmediata sobre el resto de los clientes. HTTP carece de estas características.

3.2.4 Seguridad

En cuanto a la seguridad, MQTT cuenta con autentificación de tipo identificador de cliente, usuario y contraseña o certificados x509 y con la posibilidad de controlar quién publica y quién se suscribe a cada tema. Tanto MQTT como HTTP pueden operar sobre TLS o SSL, ya que forman parte del protocolo TCP/IP, así como soportar autorización JWT (JSON Web Token). En este sentido los dos protocolos son igual de seguros y de inseguros si no se toman las medidas necesarias. En el caso de MQTT el puerto que utiliza no suele estar activado en los firewalls, por lo que es algo que también se debe tener en cuenta. En casos extremos se puede cambiar este puerto y que MQTT se implementara sobre WebSocket en el puerto 80, pero no es recomendable.

En cuanto a la seguridad implementada por cada protocolo, se puede observar que **MQTT**, **XMPP** y **HTTP** utilizan TLS para garantizar una seguridad fiable. Trabajan sobre TCP (normalmente) que no es una comunicación cifrada, así que muchos broker permiten usar TLS junto con TCP, en cambio **CoAP** utiliza DTSL para proteger la información confidencial.

TLS es el predecesor de SSL, se trata de un protocolo criptográfico que proporciona comunicación segura a través una red informática. Utiliza un mecanismo de negociación de varios parámetros para crear una conexión segura entre el cliente y el servidor.

DTLS proporciona seguridad en la comunicación de protocolos de datagramas, permitiendo a las aplicaciones basadas en datagramas comunicarse de una forma diseñada para evitar la escucha, la manipulación o la falsificación de mensajes. Inicialmente fue diseñado para dispositivos potentes conectados a través de una red fiable y con gran ancho de banda.

Ahora se van a analizar las principales características de ambos protocolos, MQTT y HTTP, según la importancia en el proyecto actual.

	MQTT	HTTP
Sencillez y ligereza (40%)	8	6
Velocidad (25%)	8	7
Fiabilidad (15%)	8	0
Seguridad (15%)	9	8
Extras (5%)	7	0
TOTAL	**8.1**	**5.35**

Tabla 6: Comparativa entre las opciones de "MQTT" y "HTTP"

3.3 ¿Es mejor usar un MQTT broker o una plataforma IoT?

Una vez se ha decidido utiliza el protocolo MQTT, ahora la pregunta es qué MQTT broker utilizar o si es mejor utilizar una plataforma IoT, se van a analizar las diferencias:

Un **MQTT broker** realiza la función esencial de la comunicación MQTT recibiendo todos los datos que son publicados por un "Publisher" en un tema concreto y los envía a los subscriptores de dicho tema. El broker es una parte fundamental en la comunicación MQTT, pero para que ésta funcione debe haber una herramienta subscrita que reciba los datos enviados por el dispositivo IoT, o en el caso contrario, una herramienta que publique los datos en el tema al que está subscrito el dispositivo IoT.

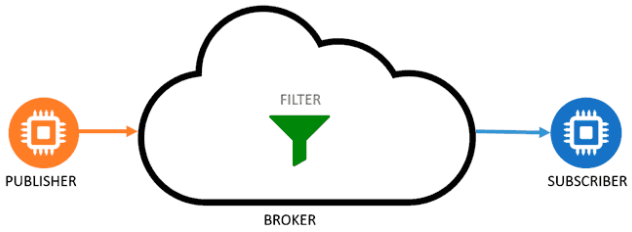

Ilustración 5: MQTT broker [4]

En el caso del internet de las cosas, estas herramientas suelen ser los "motores de procesos" (process engine en inglés). Estas herramientas se encargan de procesar, guardar y distribuir los datos y la información relacionada con un proceso. En el caso que ocupa el proyecto actual, estas herramientas pueden estar subscritas al broker y por tanto recoger los datos enviados por este y procesarlos ,o enviar estos datos al API. También pueden publicar datos en el broker y por tanto recogerlos del API y procesarlos para posteriormente publicarlos. Entre las herramientas más conocidas se encuentran Eclipse Kura, Node-red y Flogo, los tres son frameworks de código abierto, disponen de una interfaz de usuario visual para codificar, probar y depurar y se pueden conectar con tecnologías IoT a través de protocolos como MQTT, CoAP, REST, etc. [5]

Eclipse **Kura** se centra en los IoT Gateway que son dispositivos de hardware o software que sirven como punto de conexión entre la nube y los controladores, sensores y dispositivos inteligentes. [6] También se centra en el código en vez de en el diseño visual, cuenta con la implementación estándar de un broker MQTT, así como con la posibilidad de conectarse a un broker externo siendo un cliente MQTT.

Node-red fue publicado por IBM. Es un buen framework para conectar dispositivos IoT y APIs, cuenta con una interfaz de usuario con un diseño muy visual por lo que está centrado en el desarrollo a través del interfaz de usuario. Se centra también en integrar los IoT Gateway y realizar los flujos de acción. Los flujos creados se pueden compartir de una manera muy simple a través de cadenas tipo JSON. Está construido sobre JavaScript y Node.js por lo que el código de las funciones se desarrolla en estos lenguajes.

Flogo es bastante nuevo. Su interfaz de usuario es muy parecida a la de Node-red consiguiendo una implementación de código muy sencilla también. Está centrado en integrar los IoT Gateway y en los servicios de aplicaciones de borde (Edge applications en inglés) que se encargan de reducir los volúmenes de datos a mover por la red. Puede ser una buena opción si el hardware utilizado o el ancho de banda disponible es muy reducido. Como en el caso de Node-red, los flujos creados se pueden compartir de una manera muy simple a través de cadenas tipo JSON.

	Kura	Node-red	Flogo
Sencillez y curva de aprendizaje (45%)	4	9	8
Adaptación al ancho de banda reducido (25%)	6	7	9
Enfocado en IoT (20%)	7	10	8
Potencia (10%)	9	7	7
TOTAL	**5.6**	**8.5**	**8.15**

Tabla 7: Comparativa entre las opciones de "Kura", "Node-red" y "Flogo"

Por tanto, el "motor de procesos" para el broker MQTT que más encaja con el proyecto a realizar es Node-red, una herramienta sencilla, con una curva de aprendizaje muy atractiva y, sobre todo, con la potencia suficiente para el trabajo requerido.

Otra de las opciones que se han comentado es la utilización de una plataforma IoT en vez de un broker MQTT, esta plataforma es un sistema de hardware y software que permite administrar dispositivos IoT y recopilar, almacenar, visualizar y analizar datos de esos dispositivos. Algunas de las plataformas IoT que se encuentran en el mercado disponen de un MQTT broker y, además, añaden funcionalidades extras.

Se puede pensar que una muy buena opción es escoger una plataforma, pero antes de decidir qué opción escoger, se van a estudiar las más conocidas. El broker MQTT más conocido y utilizado es ***Mosquitto*** y la plataforma IoT que se va a estudiar es ***Thingsboard***.

Mosquitto se trata de un MQTT broker como se acaba de comentar, se encarga de realizar la comunicación entre los extremos de este protocolo. Se trata de un software open source publicado por Eclipse y dispone de versiones de MQTT desde la 3.1.1 hasta la 5.0 que es la más reciente, así como de todas las calidades de servicio o QoS que ofrece MQTT. Utiliza el protocolo MQTT sobre WebSokets, es decir, sobre protocolo TCP. [7] La utilización de este o cualquier otro MQTT broker ofrece la posibilidad de conocer cada paso que está realizando la comunicación entre publishers y suscriptores.

ThignsBoard se trata de una plataforma IoT cuya parte del servidor puede actuar como un broker MQTT. También ofrece la posibilidad de conectarlo a un broker MQTT externo como suscriptor para recibir los datos de los dispositivos IoT. Este tipo de plataformas ofrecen funcionalidades como recopilar datos de varias fuentes, almacenar los datos, controlar los dispositivos, mostrar los datos, ejecutar pruebas y almacenar el inventario de dispositivos, pero con ellas se obtiene una mayor abstracción en cuanto a la comunicación entre "publishers" y suscriptores.

Esta abstracción puede ser buena para aquellas personas que simplemente busquen la opción más completa, pero en el caso de este proyecto es más correcto el uso de un MQTT broker como tal, ya que de esa forma se podrá estudiar mejor las comunicaciones y el funcionamiento del protocolo MQTT.

3.4 ¿Implementar en contenedores?

Ahora es el momento de decidir donde implementar la solución escogida y una muy buena práctica es realizarlo en contenedores, pero ¿qué son?

Los **contenedores** son muy parecidos a las **máquinas virtuales**, pero tienen las siguientes diferencias principales que hacen a los contenedores ser más eficientes en pequeños trabajos y en trabajos que corren sobre el mismo sistema operativo.

Las **máquinas virtuales** virtualizan la máquina entera y trabajan sobre toda la infraestructura de ésta. Además, cada máquina virtual, tiene que cargar su propio sistema operativo gastando muchos más recursos, ya que esta instalación es igual a la que se realiza en una máquina real. La estructura de las máquinas virtuales es la siguiente: todas trabajan sobre la misma infraestructura o hardware y sobre un mismo programa que permite la instalación de estas máquinas denominado *Hypervisor*, después, cada máquina virtual debe tener, como ya se ha comentado, su propio sistema operativo (*Guest OS*) junto con nuestra aplicación y los binarios y librerías que necesite. Como se puede observar, si dos máquinas virtuales utilizan el mismo sistema operativo no pueden compartirlo, cada una debe utilizar el suyo propio. Con esto se consigue, entre muchas otras cosas, que cada máquina virtual pueda tener una versión de un programa y estas no tengan ningún problema.

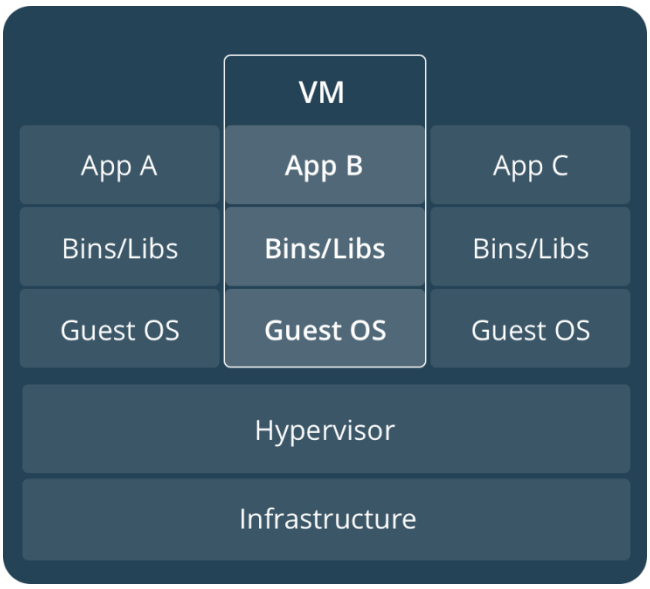

Ilustración 6: Estructura de las máquinas virtuales [8]

Los **contenedores** virtualizan el sistema operativo, con esto consiguen que cada contenedor no tenga que instalar su propio sistema operativo, si no que utilizan el kernel o núcleo del sistema operativo que tiene la máquina anfitriona. El kernel de los sistemas operativos se encarga de interactuar con el hardware, principalmente se puede encontrar el kernel de Linux sobre el que trabajan multitud de distribuciones y el kernel de Windows sobre el que trabajan todas las versiones de este sistema operativo. Por tanto, dentro de una maquina con un kernel de Linux puede haber contenedores que utilicen distintas distribuciones, pero en el caso de Windows al solo existir una distribución, lo único que se puede ver son contenedores que utilicen distintas versiones de esta distribución. Esto es una gran ventaja en el caso de los distribuidores del kernel de Linux, ya que pueden coexistir en la misma máquina sin necesidad de cargar todo el sistema operativo como en el caso de las máquinas virtuales.

Como se ha comentado, el contenedor trabaja sobre el kernel de la máquina anfitriona y se encarga de cargar las librerías, binarios y dependencias que necesitan las aplicaciones que va a albergar. Esta configuración se puede encontrar en lo que se llaman **imágenes**, estas imágenes son como plantillas que permiten la creación de contenedores. Los contenedores, aparte de lo comentado con los sistemas operativos, también permiten tener en ejecución varias versiones de un mismo programa sin la necesidad de tener en cuenta la compatibilidad. Las imágenes de los contenedores ofrecen una muy rápida configuración del sistema, ya que en un mismo archivo de configuración se encuentran los comandos necesarios para instalar las herramientas que se van a utilizar. También ofrecen la certeza de que, en todos los entornos, que dispongan únicamente del kernel necesario, la aplicación a ejecutar va a funcionar de la misma forma, ya que la configuración está realizada expresamente para que la aplicación funcione.

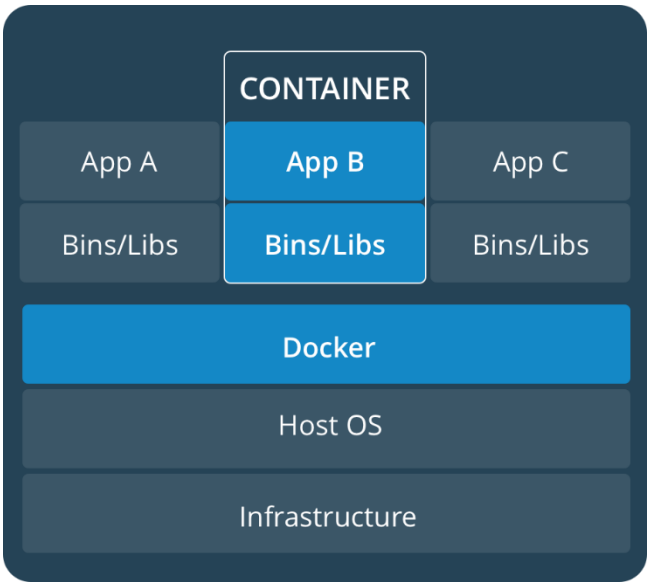

Ilustración 7: Estructura de los contenedores [8]

Docker es una plataforma de software de código abierto que permite trabajar con contenedores. Una vez se instala en una máquina, ya se pueden crear contenedores y trabajar con ellos. Es dentro del Docker donde se encuentran los contenedores, previamente creados a través de sus imágenes. Estas imágenes son muy fáciles de compartir y esta es una característica muy relevante de los contenedores, ya que la simple ejecución de esa plantilla o imagen crea el contenedor con todo lo necesario para que funcione la aplicación que alberga.

Otra de las características más relevantes de los contenedores o más en concreto de Docker, es que trabaja con una arquitectura de capas, es decir, cada configuración que aparece en la imagen o plantilla crea una capa nueva, por tanto, dos imágenes que tengan la misma configuración inicial compartirán esas capas, así como cuando una imagen sea modificada, si las primeras capas no lo son, estas no tienen que volverse a crear. Con esto se consigue optimizar el espacio que ocupan y el tiempo que se tarda en reconstruir o actualizar una imagen.

En el caso que ocupa el proyecto actual, no es tan relevante la posibilidad o no de correr aplicaciones que utilicen diferentes sistemas operativos o diferentes versiones de un mismo programa, si no la posibilidad de compartir las imágenes de los contenedores y la optimización de recursos lo que los hace más interesantes.

	Implementación en host	Máquina Virtual	Contenedor Docker
Capacidad de compartir (35%)	0	0	10
Compatibilidad con cualquier sistema operativo (25%)	0	10	5
Facilidad de implementación (20%)	10	10	6
Optimización de recursos (10%)	10	3	9
Facilidad de instalación (10%)	4	6	10
TOTAL	**3.4**	**5.4**	**7.85**

Tabla 8: Comparativa entre las opciones de "Máquina Virtual", "Contenedor o Docker" e "Implementación directa en un host local"

Otra de las ventajas de los contenedores, es que ya existen muchas imágenes creadas por usuarios en el Docker Hub [9] donde se describen los comandos necesarios para la instalación de múltiples programas, por lo que la simple ejecución de esa imagen carga el programa. En el caso de este proyecto, existen imágenes tanto de *Mosquitto*, como de *Node-red* dentro del Docker Hub, por lo que para instalarlos solo se tiene que crear un contenedor con esas imágenes. En caso de querer modificar una imagen o añadirle configuraciones extra, solo se tendrá que crear una propia imagen basándose en las que ya existen en Docker Hub. Esto ahorra mucho tiempo y con ello se consigue tener con unas simples líneas la configuración exacta que necesita la aplicación.

3.5 ¿Cómo comunicar el servidor con la base de datos?

La forma más común de conectar las aplicaciones con la base de datos es mediante consultas SQL, es decir, directamente desde la aplicación se realiza una consulta SQL sobre la base de datos para insertar, actualizar, eliminar o consultar registros. Es la forma más sencilla y con ella se consigue una conexión mucho más directa y rápida con la base de datos. Pero uno de los requisitos para que esta forma de conexión funcione, es que la base de datos y las aplicaciones se encuentren en la misma red y el mismo servidor, ya que, a través de simples consultas SQL la aplicación debe poder comunicarse con la base de datos. En el proyecto actual esto no ocurre, ya que se cuenta con un servidor donde se alojan las aplicaciones y otro donde se encuentra la base de datos, por lo que esta no es accesible desde las aplicaciones. Esto ocurre porque el propósito de este proyecto es conseguir una integración entre dispositivos IoT y las implementaciones software de las que disponen los clientes de la empresa i3Code y estas implementaciones tienen una estructura concreta y las bases de datos se encuentran en un servidor concreto de la empresa. Por otro lado, realizando otro tipo de conexiones se consigue que cualquier base de datos sea accesible esté o no en el mismo servidor que las aplicaciones y así conseguir una implementación mucho más versátil y que pueda servir para muchos más clientes.

Otra de las opciones de conexión entre las aplicaciones y la base de datos es SOA. SOA es una arquitectura de software la cual se basa en la integración de aplicaciones mediante servicios. Un servicio es una unidad autónoma de una o más funciones software diseñada para realizar una tarea específica, como recuperar cierta información o ejecutar una operación. SOA expone los servicios utilizando protocolos estándar de red para enviar solicitudes o acceder a los datos, SoAP y JSON entre otros, por lo que no es necesario que los desarrolladores realicen las integraciones desde cero. [10] El servidor puede utilizar tanto SoAP como REST para conectarse con la base de datos.

SoAP es un protocolo para el manejo de WebServices el cual está basado en XML y su objetivo es la comunicación entre servicios que se exponen por contrato. Se trata de un protocolo que define cómo dos objetos en diferentes procesos se comunican mediante mensajes XML. Fue creado por un grupo de empresas entre las que destacan Microsoft e IBM, pero actualmente lo gestiona W3C, un consorcio internacional que genera estándares para la Web.

REST es un protocolo para el manejo de WebServices el cual trabaja sobre HTTP. Puede manejar una comunicación basada en XML y JSON, exponer URIs y manejar si se requiere un contrato similar al protocolo SoAP (WDSL), que es conocido como WADL. Se trata de una arquitectura ligera muy utilizada en la comunicación de aplicaciones tipo móvil e IoT, emplea el protocolo HTTP y proporciona una API para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio web y el cliente. Esta API ofrece métodos de interacción como GET, POST, PUT y DELETE entre otros.

RESTful es un programa que se encarga del manejo de WebService pero aplicando en su comunicación el protocolo REST. Los servicios Web RESTful cuentan con cuatro operaciones que son GET (listar un conjunto de objetos o leer un objeto concreto, dependiendo de cómo se cree la URI), POST (crear), PUT (actualizar) y DELETE (eliminar). Cada operación requiere de un tipo de parámetros y por tanto de un método URI distinto.

3.5.1 Extensión del mensaje

Una de las principales diferencias entre SoAP y REST es que el primero es una implementación de SOA basada en XML y diversos protocolos, en cambio, REST, aunque es también una implementación de SOA, está basado en JSON y estrictamente HTTP. Esto hace que la extensión de la misma cantidad de datos en ambos protocolos sea diferente. En el caso de SoAP es obligatorio envolver el fichero XML con SoAP Envelope, una etiqueta que define qué hay en el mensaje y cómo se debe procesar, además el fichero XML cuenta con una cabecera y el cuerpo del mensaje donde ya vienen los datos a enviar. En el caso de REST, al utilizar JSON, es directamente en el mensaje donde aparecen los datos, sin cabeceras de ningún tipo, por lo que la extensión en este segundo protocolo es bastante más reducida.

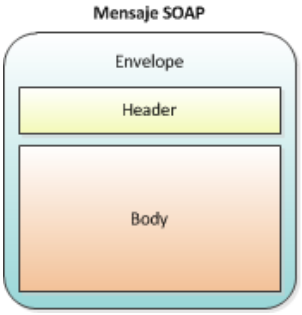

Ilustración 8: Estructura de un mensaje SoAP [11]

3.5.2 Número de URLs

El número de URLs necesarias para la comunicación también es una de las diferencias existentes entre ambos protocolos. Mientras que SoAP solo necesita una única URL añadiendo los parámetros necesarios mediante petición POST, REST necesita una petición por cada acción y operación a realizar. Por ejemplo, para tratar con los dispositivos se deben tener la siguientes URLs.

Operación	URL	Definición
GET	GET/dispositivos	Acceder al listado de los dispositivos
POST	POST/dispositivos	Crear un dispositivo
GET	GET/dispositivos/id_dispositivo	Acceder al detalle del dispositivo con el id dado
PUT	PUT/dispositivos/id_dispositivo	Actualizar el dispositivo con el id dado y sustituyendo toda su información anterior por la nueva
DELETE	DELETE/dispositivos/id_dispositivo	Eliminar el dispositivo con el id dado

Tabla 9: URLs necesarias en el protocolo REST

3.5.3 Documentación

Otra de las principales diferencias es la cantidad de documentación que se realiza en los protocolos. En el caso de SoAP la documentación es una parte muy importante del protocolo ya que siempre cuenta con un fichero de descripción del lenguaje *.WSDL* ya que es parte de su estándar. En cambio, en el caso de REST, en el 90% de los casos no se tiene un fichero de descripción del lenguaje, sino que se dan ejemplos de cómo utilizar la API. En caso de ofrecer esta documentación, no existe una estandarización y cada proyecto puede tener su propio tipo de archivos, aunque entre todos ellos destacan Swagger y RAML.

3.5.4 Usos

Por último, se encuentran con la diferencia de los entornos en los que se suele utilizar cada protocolo. Como se ha visto, SoAP es un protocolo más pesado, sus mensajes ocupan más y necesita bastante documentación. Fue creado en 1998 por Dave Winer con la colaboración de

Microsoft [12] y está diseñada principalmente para las necesidades del mercado de las empresas. Por tanto, el principal entorno de SoAP es el entorno empresarial, dedicándose a las comunicaciones transaccionales entre compañías que están asociadas. Con la existencia siempre de un archivo .WSDL con el acuerdo de especificación de campos, se obtiene un fácil control de cambios y versionados además de una facilidad de testeo y obtención de trazas en temas de seguridad gracias a WS-Security. En el caso que ocupa este proyecto no es muy recomendable el uso de SoAP ya que no obtiene un buen rendimiento en aplicaciones web, móvil e IoT ya que solo utiliza el backend.

En cambio, REST es mucho más liviano, sus mensajes ocupan menos y no requiere de apenas documentación. Fue creado en 2000 por Roy Fielding en el ambiente académico [12] con la filosofía de ser usado en aplicaciones web abiertas. Por tanto, el principal entorno de REST es la comunicación entre partes de un mismo sistema, por ejemplo, Webs, Aplicaciones móvil o arquitecturas de microservicios. Es un protocolo muy popular entre los programadores ya que es desarrollado en web y JS, no requiere de formación adicional y es muy fácil ponerlo en marcha. Además, obtiene un buen rendimiento en aplicaciones web, móvil e IoT ya que utiliza tanto el backend como el frontend, algo muy importante para el proyecto actual.

Conociendo estas diferencias es momento de evaluar lo más importante de cara al proyecto a realizar.

	SoAP	REST
Rendimiento en IoT (50%)	3	10
Mensajes reducidos (25%)	4	10
Facilidad de uso (15%)	5	9
Documentación no exhaustiva (10%)	3	9
TOTAL	**3.55**	**9.75**

Tabla 10: Comparativa entre los protocolos "SoAP" y "REST"

Tras evaluar las características más importantes en el proyecto se puede concluir que el protocolo que mejor valoración ha obtenido es el protocolo REST, por tanto, es el que se va a desarrollar.

3.6 ¿Qué posibilidades ofrece Azure?

Existen multitud de plataformas que ofrecen servicios donde desplegar aplicaciones en la nube, pero las más potentes actualmente en el mercado son: **AWS** y **Microsoft Azure**. i3Code utiliza servidores de Azure ya que se trata de una empresa que trabaja con todo el software de Microsoft así que se va a analizar los servicios que ofrece.

Microsoft Azure ofrece multitud de herramientas, recursos y soluciones, pero el estudio, se va a centrarnos solamente en aquellas que pueden ser útiles para el actual proyecto.

3.6.1 Máquinas Virtuales

Una de las posibilidades que ofrece Azure son las **Máquinas Virtuales**, estas máquinas proporcionan la flexibilidad que ofrece la virtualización para una amplia gama de soluciones de computación: opciones de desarrollo y pruebas, ejecución de aplicaciones y ampliación del centro de datos. La configuración del software de código abierto proporciona libertad ya que es como tener otro bastidor en el centro de datos. Azure Virtual Machines admite IBM, Oracle, Red Hat, SAP, SQL Server, Linux y Windows Server. [14] Además, da la posibilidad de llevar un registro de los gastos de las máquinas virtuales y ofrece una previsión de costes a largo plazo.

Esta máquina virtual ofrece una IP pública a través de la cual se puede acceder al servidor. Se puede alojar un servidor Web dentro de la máquina virtual y será accesible a través de la IP publica dada o a través del nombre de dominio o DNS que Azure permite añadir a la máquina virtual. Estas máquinas virtuales vienen solo con el sistema operativo escogido y en ellas se puede instalar cualquier cosa. Por ejemplo, se puede crear una máquina virtual con Linux e instalar Docker para poder alojar contenedores en dicha máquina virtual y que estén accesibles desde la red global.

3.6.2 Contenedores

En cuanto al manejo de contenedores, también dispone de varias herramientas que es interesante comentar.

Azure pone a disposición de sus clientes la siguiente tabla con la que conocer, de forma muy general, el uso a cada una de esas herramientas o recursos en cuanto a contenedores:

SI QUIERE...	UTILICE...
Simplificar la implementación, la administración y las operaciones de Kubernetes	Azure Kubernetes Service (AKS)
Cree eficaces aplicaciones en la nube con rapidez para la Web y móviles	App Service
Ejecute contenedores en Azure fácilmente sin administrar servidores	Container Instances
Habilite la nube para la programación de trabajos y la administración de procesos	Batch
Desarrolle microservicios y organice contenedores en Windows o Linux	Service Fabric
Almacene y administre imágenes de contenedor en todos los tipos de implementaciones de Azure	Container Registry
Ejecute clústeres de OpenShift totalmente administrados que se ofrecen en colaboración con Red Hat	Red Hat OpenShift en Azure

Ilustración 9: Tabla de los recursos de contenedores de Azure [15]

3.6.2.1 Servicio de Kubernetes AKS

Se va a comenzar según el orden de la tabla por el servicio de Kubernetes AKS y primero de todo se debe conocer que es Kubernetes.

Kubernetes es una orquestación que consiste en un conjunto de herramientas y scripts que pueden ayudar al ingeniero de sistemas que debe mantener el orden y correcto funcionamiento de los contenedores de un host en el entorno de producción. Normalmente son

múltiples Docker Host que pueden contener contenedores, por lo que, si uno falla, se puede acceder a las aplicaciones a través del resto. Kubernetes cuenta con varios tipos de herramientas para la ejecución de contenedores como containerd, CRI-O y Docker, aunque el más común es este último.

Kubernetes es la orquestación más popular, es un poco complicada de desplegar, pero ofrece muchas opciones para customizar el despliegue. Permite crear cientos de instancias de una misma aplicación y escalar el número de instancias gracias al Kubernetes CLI (línea de comandos) denominada "cube control". Incluso se puede configurar todo esto de forma automática para que Kubernetes escale el número de instancias según el uso de los usuarios, así como actualizar estas múltiples instancias de la imagen mediante una actualización y sí algo va mal se puede retroceder esa actualización o testear las nuevas versiones actualizando solo parte de las instancias que se tienen en ejecución. Un clúster de Kubernetes consiste en un conjunto de nodos. Estos nodos son máquinas de tipo worker, físicas o virtuales, donde están instaladas una serie de herramientas que les permiten almacenar contenedores. El clúster consigue que, si un nodo falla, la aplicación no se quede incomunicada y falle también, si no que pueda estar accesible a través del resto de nodos. Kubernetes cuenta también con un nodo Máster el cual observa el funcionamiento de los nodos y se encarga de distribuir los servicios entre ellos de la mejor forma.

En el caso del servicio de Kubernetes AKS de Azure, utiliza la herramienta Docker para la ejecución de los contenedores. AKS ofrece un servicio de Kubernetes administrado, sin servidor y con integración y entrega continuas (CI/CD). AKS permite definir, implementar, depurar y actualizar fácilmente cualquier aplicación de Kubernetes e incluirlas en contenedores de forma automática. También ofrece la posibilidad de agregar una canalización de CI/CD completa a los clústeres con automatización de tareas rutinarias y configurar una estrategia de implementación controlada. Azure asegura que se pueden detectar errores de forma temprana y optimizar la canalización con una rastreabilidad exhaustiva. Ofrece una eficiencia operativa a través del aprovisionamiento automatizado, la reparación, la supervisión y el escalado integrados.

Algunos de los escenarios más comunes en los que se utiliza AKS son:

1. **Migrar mediante "lift-and-shift" a contenedore**: migrar fácilmente las aplicaciones a contenedores y ejecutarlas de forma sencilla en AKS.
2. **Microservicios**: ASK simplifica la implementación y administración de una arquitectura basada en servicios.
3. **DevOps seguro para AKS:** DevOps ofrece seguridad y Kubernetes velocidad.
4. **Expansión instantánea de AKS con ACI**: el nodo virtual de AKS puede aprovisionar de contenedores a ACI.
5. **Arquitectura de referencia de Azure IoT**: para aplicaciones que usan componentes PaaS
6. **Entrenamientos de modelos de Machine Learning**: AKS ofrece herramientas conocidas para el aprendizaje de modelos mediante grandes conjuntos de datos.
7. **Escenario de streaming de datos**: AKS permite ingerir y procesar datos en tiempo real.

En conclusión, esta herramienta que ofrece Azure es interesante para proyectos grandes o con visión de futuro por el hecho de poder de forma automatizada escalar los recursos de las aplicaciones en segundos, entre otras cosas.

Para el proyecto actual, puede ser una herramienta interesante en cuanto a las líneas futuras. Por ejemplo, si se diera el caso de que se contara con muchas empresas asociadas a la misma aplicación y con muchos usuarios consultando los datos recibidos, o con muchos

dispositivos recabando información y mandándola para que se almacene en la base de datos y se muestre en la página web, se podría necesitar ese escalado y de esa administración de contenedores con detección de errores de forma temprana.

3.6.2.2 Servicio de aplicaciones

Los servicios de aplicaciones o *App Service* de Azure son una plataforma para crear, implementar y escalar aplicaciones web. Esta herramienta permite trabajar con .NET, .NET Core, Node.js, Java, Python o PHP, bien en contenedores o en ejecución en Windows o Linux. Azure ofrece crear APIs y aplicaciones web en la nube simplemente con el código en cualquier lenguaje, aumentando la productividad debido a la integración con plataformas de desarrollo, optimizando CI/CD con diversas plataformas de control de versiones como son Git, GitHub, Acciones de GitHub, Atlassian Bitbucket, Azure DevOps, Docker Hub y Azure Container Registry.

Azure permite modificar la escala de las aplicaciones ofreciendo la posibilidad de conseguir alta disponibilidad, simplificando las operaciones con la automatización del mantenimiento de la plataforma y la aplicación de revisiones de seguridad, proteger las aplicaciones en *Web App Firewall*, implementar instancias aisladas de aplicaciones web con un modelo de inquilino único, usar *Azure Active Directory* u otro proveedor de identidades para autentificar el acceso y ampliar la escala global a todas las regiones de Azure. Además, Azure dispone de *Azure App Service Migration Assistant*, una herramienta sencilla y gratuita para migrar automáticamente aplicaciones web de .NET del entorno local a la nube.

Azure ofrece la posibilidad de innovar con los servicios cognitivos y basados en eventos de Azure usando *App Service* para incorporar *Cognitive Services*, que mejora la accesibilidad insertando lectura de textos y traducción de voz a las aplicaciones o *Personalizer* la inteligencia artificial de Azure que puede incorporar búsquedas o facilitar el uso de la aplicación. También permite simplificar las operaciones con la supervisión integrada. Es capaz de solucionar los problemas en vivo de forma inteligente e interactiva con los diagnósticos de *App Service*, así como supervisar el rendimiento y el estado de mantenimiento de las aplicaciones de forma integral o tomar las decisiones con mayor rapidez a través de *Azure Monitor* y *Application Insights*. Con *Azure Monitor* también se pueden crear vistas en tiempo real del uso o configurar alertas entre otras cosas. *Application Insights* permite además obtener información más detallada sobre el rendimiento de la aplicación.

En conclusión, esta herramienta de Azure ayuda y facilita el despliegue de aplicaciones web a APIs en la web. Es una herramienta muy interesante y que ofrece muchas posibilidades.

En cuanto al proyecto actual, esta parte del desarrollo viene definida por la empresa i3Code. El proyecto les va a servir para poder proponer a sus clientes soluciones IoT integradas con las aplicaciones que ya disponen y han sido creadas también por ellos. Por esto misma la implementación de esta parte del proyecto debe ser lo más similar a las aplicaciones que despliega i3Code con sus clientes, pero el hecho de conocer esta herramienta y lo fácil que es migrar automáticamente aplicaciones web de .NET del entorno local a la nube a través de Azure *App Service Migration Assistant*, es relevante para i3Code ya que desarrolla sus aplicaciones en este lenguaje.

3.6.2.3 Instancias de contenedores

A través de las instancias de contenedores o ACI, Azure da la posibilidad a sus clientes de desarrollar aplicaciones de forma rápida sin necesidad de administrar máquinas virtuales ni otras herramientas ya que la aplicación se ejecutará en la nube dentro de un contenedor. Debido a que no es necesario dedicarse a la administración de la infraestructura en la que se ejecutan

las aplicaciones, Azure asegura que los usuarios pueden dedicarse al diseño y la creación de esas aplicaciones. Además, Azure ofrece la posibilidad de usar ACI para aprovisionar cálculos adicionales para cargas de trabajo exigentes cuando sea necesario. Por ejemplo, con *Virtual Kubelet*, se puede utilizar ACI para ampliar elásticamente en ráfagas desde el clúster de AKS cuando haya picos de tráfico. ACI ofrece aislamiento de hipervisor para cada grupo de contenedores con el fin de garantizar que los contenedores se ejecutan de forma aislada sin compartir un kernel.

Virtual Kubelet es una implementación de Kubelet de Kubernetes que se hace pasar por un Kubelet con el fin de conectar un clúster de Kubernetes a otras API. Esto permite que los nodos de Kubernetes estén respaldados por otros servicios, como las plataformas de contenedores sin servidor.

En cuanto a lo que se puede crear con ACI, se encuentra la ampliación elástica en ráfagas con AKS, ya que ACI ofrece cálculos aislados y rápidos para atender los picos de tráfico sin tener que administrar servidores. También se pueden crear aplicaciones basadas en eventos con *Azure Logic Apps*, ya que con ACI se pueden ejecutar tareas complejas capaces de dar respuesta a los eventos. Así como la creación de trabajos de procesamiento de datos cuando los datos de origen se ingieren, procesan y colocan en un almacén duradero tal como *Azure Blob Storage*, ya que al procesar los datos con ACI en lugar de máquinas virtuales aprovisionadas estáticamente, puede conseguir un importante ahorro de costos.

En concusión, la herramienta ACI es una herramienta que facilita el uso de contenedores y sus instancias, permitiendo a sus usuarios dedicarse al diseño de las aplicaciones.

En cuanto al proyecto actual, según las posibles implementaciones de ACI que expone Azure es su página web no considero que pueda exprimir todo lo que ofrece, por lo que no considero que aporte mucho valor al proyecto esta herramienta. Además, ACI está destinado a escenarios efímeros o que no necesitan ejecutar contenedores de forma permanente, como es este caso en el que se necesita tener una conexión continua entre el dispositivo y el api que cargará esos datos en la base de datos. Aun así, esta herramienta es muy útil para los escenarios efímeros, que requieren de muchos recursos durante un tiempo limitado, ya que ACI se encargará de gestionar los recursos y el cliente pagará solo por el tiempo de uso.

3.6.2.4 Batch

Es muy común encontrarse con procesos lote que requieren un tiempo mayor a otras tareas del negocio: facturación, renderización de imágenes, generación de informes, etc. Son ese tipo de tareas que usualmente se realizan por la noche porque llevan tiempo de procesado o necesitan de alto rendimiento de computación. Para esto está pensado *Azure Batch*, que permite de manera programática definir un conjunto de recursos en Azure encargados de ejecutar trabajos bajo demanda o programados, sin la necesidad de configurar un clúster HPC, máquinas virtuales, etc.

Por tanto, *Azure Batch* ofrece posibilidad de escalar multitud de máquinas virtuales, habilitar la nube en aplicaciones HPC y de lote, realizar copias intermedias de los datos, ejecutar canalizaciones de proceso, elegir entre Linux o Windows para ejecutar trabajos, aplicar escalado automático según el trabajo en cola y conseguir potencia de computación por lotes cuando se necesite. El procesamiento por lotes se inició con los equipos informáticos centrales y las tarjetas perforadas.

Azure Batch permite elegir el sistema operativo y las herramientas de desarrollo que se necesiten para ejecutar trabajos a gran escala ofreciendo una experiencia coherente de administración y programación de trabajos permitiendo a sus clientes aprovechar las características únicas de cada entorno. En *Azure Batch* se encuentra SDK y compatibilidad con diversas herramientas de desarrollo, como Python y Java. Además, ofrece la posibilidad de tener acceso a múltiples núcleos para ejecutar el trabajo y pagar solo por los que se utilizan. El núcleo de *Azure Batch* es un motor de programación de trabajos a gran escala, que los usuarios tienen a su disposición como servicio administrado. Al ejecutar un trabajo, *Azure Batch* iniciará un grupo de máquinas virtuales de proceso, instalará aplicaciones y realizará copias intermedias de los datos, ejecutará trabajos con tantas tareas como tenga, identificará errores y volverá a poner el trabajo en cola, y reducirá verticalmente el grupo cuando el trabajo haya concluido. Además, procesa trabajos a petición, no según una programación predefinida, por tanto, los clientes ejecutan trabajos en la nube cuando necesitan hacerlo, y se puede administrar que usuarios puede tener acceso y cuántos recursos van a poder utilizar, gracias la exhaustividad de los informes se puede realizar un seguimiento del uso.

La última funcionalidad que ofrece *Azure Batch* es la representación de animaciones, ya que permite ejecutar trabajos de representación en la nube evitando la complejidad de administración y reduciendo el tiempo necesario para realiza los trabajos.

En conclusión, *Azure Batch* es una herramienta pensada para ejecutar aquellos procesos de lotes que necesitan de muchos recursos o de tiempo de ejecución. Además de esto ofrece multitud de herramientas para gestionar estos lotes. En cuanto al proyecto actual no es una herramienta que pueda ser útil ya que no se va a ejecutar ese tipo de procesos, sino que se va a ejecutar, como se ha comentado antes, un proceso continuo que necesita pocos recursos y debe estar activo de forma persistente.

3.6.2.5 Service Fabric

Azure Service Fabric es un proyecto de código abierto diseñado para ofrecer servicios duraderos con alta disponibilidad a escala de nube, comprende intrínsecamente la infraestructura disponible y las necesidades de recursos de las aplicaciones, lo que permite escalado automático, actualizaciones graduales y recuperación automática cuando se producen errores. Sustenta la infraestructura principal de Azure y otros servicios de Microsoft, como Skype Empresarial, Intune, Azure Event Hubs, Azure Data Factory, Azure Cosmos DB, Azure SQL Database, Dynamics 365 y Cortana.

Azure Service Fabric permite elegir entre una gran variedad de lenguajes y modelos de programación productivos, como .NET Core 2.0, C# y Java, para crear aplicaciones basadas en microservicios y contenedores, implementar el clúster de *Azure Service Fabric* en Azure o usar *Azure Service Fabric mesh*, una plataforma de microservicios completamente administrada que actualmente está en versión preliminar. *Azure Service Fabric* también está disponible como descarga gratuita para Windows Server, lo que permite crear clústeres de *Azure Service Fabric* de forma local o en otras nubes.

Azure Service Fabric asegura la simplificación de la creación y administración de aplicaciones de microservicios y es que Azure se encarga de cuestiones de confiabilidad, escalabilidad, administración o latencia de infraestructuras permitiendo a sus usuarios centrarse en características que aporten más valor comercial a sus aplicaciones, además, cuenta con características de hospedaje de contenedores, administración de recursos de clúster y organización de cargas de trabajo. Con todo esto se consigue una rápida comercialización y una

organización de contenedores y servicios. Azure, permite a sus usuarios combinar los lenguajes y modelos de programación que mejor se adapten a sus necesidades, desde contenedores y archivos ejecutables invitados, hasta microservicios y actores, así como integrar IDEs como Visual Studio, Eclipse o la línea de comandos compilando, probando, depurando, implementando y actualizando aplicaciones de forma rápida y sencilla en implementaciones de prueba y de producción. Uno de los puntos fuertes de *Azure Service Fabric* es que permite la ejecución de aplicaciones en cualquier parte, implementando con flexibilidad el mismo código de aplicación en nubes públicas, hospedadas o privadas usando servicios de plataforma coherentes y los mismos modelos de programación de aplicaciones, con la opción de elegir Windows Server o Linux como sistema operativo host.

Azure Service Fabric asegura a sus clientes que podrán entregar actualizaciones rápidas y seguras sin tiempo de inactividad, automatizar operaciones a escala e integrar supervisión del estado y recuperación automática en caso de errores. Además, ofrece la posibilidad de organizar aplicaciones basadas en microservicios y contenedores, conociendo el estado y el rendimiento de las aplicaciones y habilitando el desarrollo de servicios resistentes, con baja latencia y capacidad de escalado. *Azure Service Fabric* afirma que sus usuarios podrán solucionar los complejos problemas de los sistemas distribuidos y ofrecer características de administración del ciclo de vida de las aplicaciones para que los desarrolladores no tengan que rediseñar las aplicaciones conforme aumenta su uso, al mismo tiempo que desarrollar y entregar muchos tipos de aplicaciones y cargas de trabajo.

En conclusión, Service Fabric es una plataforma de sistemas distribuidos que facilita el empaquetado, despliegue y gestión de microservicios escalables y confiables, teniendo en cuenta los desafíos de desarrollar y administrar aplicaciones en la nube.

En cuanto al proyecto actual, al no tratarse el mismo de una plataforma de sistemas distribuidos, no se cree que sea la herramienta de Azure más adecuada, además, parece que de las herramientas que se va a comentar, es la que menos relación con los contenedores tiene.

3.6.2.6 Registro de contenedores

El registro de contenedores o *Azure Container Registry* es un registro privado que ofrece Azure a sus usuarios. Es muy común compartir imágenes a través del repositorio público de Docker Hub, sin embargo, las empresas en muchas ocasiones requieren de un repositorio privado. Para esto hay diferentes opciones, como el registro privado propio de Docker Hub, crear un repositorio local y el registro de contenedores privado de Azure. *Azure Container Registry* es capaz de establecer conexiones entre entornos, incluyendo AKS y *Red Hat OpenShift* en Azure, así como servicios de Azure, como *App Service*, *Machine Learning* y *Batch*.

Azure ofrece este servicio para que sus usuarios simplifiquen la administración del ciclo de vida de los contenedores creando, almacenando, protegiendo, examinando, replicando y administrando imágenes de contenedores. Además de almacenar las imágenes, *Azure Container Registry* es capaz de controlar formatos de contenido relacionados con ellas como son los gráficos Helm, los artefactos OCI e imágenes creadas conforme a la especificación de formatos de imagen OCI. *Azure Container Registry* también ofrece la posibilidad de optimizar la creación, las pruebas, la inserción y la implementación de imágenes en Azure con *Azure Container Registry Tasks*. Por ejemplo, se pueden crear tareas para recompilar automáticamente imágenes de aplicación cuando se actualicen las imágenes base, o bien automatizar la compilación de imágenes cuando el equipo haga “commit” del código en un repositorio de Git. También se pueden crear tareas de varios pasos para automatizar la compilación, las pruebas y la aplicación

de revisiones de varias imágenes de contenedor en paralelo en la nube. *Azure Container Registry* ofrece la posibilidad de escalar los recursos globalmente creando un único registro para atender usuarios y hosts con replicas geográficas. Se puede agilizar la implementación usando webhooks específicos de cada región para recibir notificaciones cuando haya disponible una incorporación de cambios local.

Como ya se ha comentado, *Azure Container Registry* brinda a sus usuarios la seguridad de una red privada y la solidez de un servicio administrado con replicación geográfica usando *Azure Virtual Network* y reglas de firewall. Además, se puede restringir el acceso a un registro de los servicios implementados en redes virtuales, como puede ser una instancia en AKS. Otra de las características de *Azure Container Registry* es la posibilidad de firmar las imágenes que se insertan en los registros y configurar los consumidores de imágenes para que comprueben la integridad de las imágenes al extraerlas, asegurando así de que el contenido que extrae del registro es el contenido que se ejecuta en el nodo. Con la ayuda de *Azure Security Center* se pueden examinar las imágenes de *Azure Container Registry* detectando posibles vulnerabilidades conocidas en los paquetes u otras dependencias definidas en el archivo de imagen del contenedor, además, los usuarios pueden recibir valoraciones y recomendaciones sobre las vulnerabilidades incluyendo indicaciones para solucionarlas.

En resumen, *Azure Container Registry* es un repositorio privado de imágenes que ofrece una serie de herramientas para administrar las imágenes, mantenerlas seguras y detectar problemas en ellas. Además, ofrece un escalado de recursos que a priori parece sencillo. Esto último no es muy relevante para el proyecto actual, pero el hecho de poder almacenar las imágenes de forma privada y compartirlas al mismo tiempo que mantenerlas seguras y sin vulnerabilidades sí que puede ser de importancia para futuros pasos del proyecto actual.

En el caso en el que la empresa i3Code quiera implementar la solución obtenida en este proyecto en otros clientes o incluso comercializar con ella como un producto propio, el hecho de disponer de las imágenes de forma privada y segura es muy relevante, porque si fueran accesibles desde repositorios públicos, como Docker Hub, perderían todo el valor comercial. Y por otor lado el hecho de saber que desde Azure se comprueban estas imágenes para que no tengan problemas es un punto a favor y de seguridad a la hora de desplegarlas o compartirlas como un trabajo propio estando seguros de que no van a mostrar grandes vulnerabilidades.

3.6.2.7 Red Hat OpenShift en Azure

Red Hat es una multinacional de software que provee software de código abierto principalmente a empresas. OpenSift es una distribución de Kubernetes creada por Red Hat. Está centrada en la experiencia del desarrollador y la seguridad de las aplicaciones de forma independiente a la plataforma. OpenSift ayuda a sus usuarios a desarrollar e implementar aplicaciones en uno o más hosts, estas aplicaciones pueden ser aplicaciones web de frontend o aplicaciones de backend incluyendo microservicios o bases de datos.

Una vez conocidas las herramientas, Azure ofrece un servicio OpenSift totalmente administrado, es decir, da la posibilidad de crear, implementar y escalar aplicaciones OpenSift con confianza. Azure permite a sus usuarios centrarse en el desarrollo de las aplicaciones mientras Microsoft y Red Hat se ocupan de actualizar y supervisar los nodos maestros, de infraestructura y de aplicación, así como de aplicarles las revisiones necesarias. El usuario puede elegir sus propias soluciones de registro, redes, almacenamiento o CI/CD, o bien, utilizar soluciones integradas que cuentan con administración automatizada del código fuente, compilaciones de aplicaciones y contenedores, implementaciones, escalado, etc.

Azure ofrece seguridad con el inicio de sesión integrado en *Azure Active Directory* y puntos de conexión de clúster privados, así como la operaciones y seguridad a nivel empresarial ejecutando las aplicaciones en la plataforma de Kubernetes reforzada con seguridad de Red Hat OpenShift basada en la infraestructura de talla mundial de Azure. Permite a sus usuarios promover la productividad con canalizaciones de CI/CD integradas que proporcionan una amplia gama de tecnologías admitidas y flujos de trabajo optimizados, conectando después las aplicaciones a cientos de servicios de Azure como *Azure Database for MySQL*, *Azure Redis Cache* y *Azure Cosmos DB*. Otro punto importante es la posibilidad que ofrece de escalar los recursos cuando el usuario necesite, puede crearse un clúster de alta disponibilidad y escalarlo después a medida que lo requiera. Cuenta con nodos de uso general, optimizados para memoria u optimizados para procesos.

En mi opinión este recurso si está más pensado para empresa y proyectos grandes que no es el caso de este proyecto. Está bien conocer esta herramienta que ofrece Azure porque en algún momento puede ser de utilidad, pero no es el caso el proyecto ya que este no va a contar con la distribución OpenSift de Red Hat, ni con gran número de nodos trabajando.

4. Sistema

El sistema de este proyecto al completo estaría descrito de la siguiente manera:

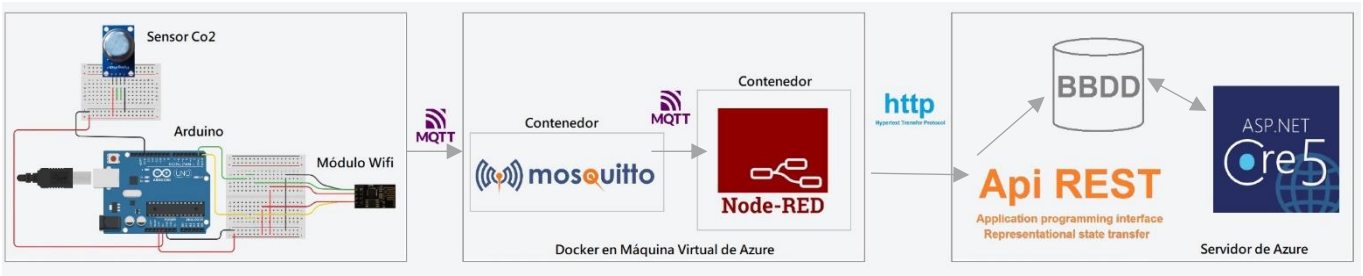

Ilustración 10: Imagen del sistema a desarrollar

El sistema se compone de dos partes, una relacionada con la comunicación y la red y otra relacionada con la implementación software. La primera parte se compone tanto del componente hardware del sistema, como de la comunicación entre los componentes, en cambio, la segunda parte se compone del desarrollo del software final. Esta segunda parte es muy parecida a la implementación que disponen los clientes de la empresa i3Code.

A continuación, se va a comentar con mayor detalle cada parte del sistema.

4.1 Parte de red

La parte de red se encarga de recoger los datos relacionados con la calidad del aire, a través del componente hardware y enviarlos a través de MQTT a un MQTT broker. Con esto se accede a la parte de comunicación del sistema que se basa en trasmitir esta información desde este MQTT broker hasta la implementación software, más concretamente, hasta la base de datos.

4.1.1 Componente hardware

El componente hardware, cuenta con un Arduino UNO R3 de ELEGOO, un módulo wifi ESP8266 ESP-01 y un sensor de gases y aire MQ135. Se han escogido estos componentes debido a que se han considerado los más aptos y mejor valorados para los requisitos del proyecto actual como se expresa en la parte de Estado del arte.

Para el montaje se ha atendido a las documentaciones de los módulos dónde se puede encontrar la forma correcta para su conexión al Arduino, así como pequeños programas sencillos para conocer su funcionamiento. Esos ejemplos se han utilizado para probar que los módulos funcionaban correctamente, comprender su funcionamiento y así poder crear el programa perfecto para este proyecto. Este programa debe recoger los datos de la calidad del aire a través del sensor MQ135 y mandarlo al MQTT broker a través del módulo wifi ESP8266 ESP-01. Se trata de un programa sencillo pero que solo puede probarse por completo si el resto del sistema está completo, o al menos si lo está el MQTT broker. El montaje se puede ver en el Anexo 1.

Para que el sensor MQ135 recoja los datos de la calidad del aire es necesario, como indica el fabricante, calibrarlo. Esta calibración se detalla en el Anexo 2. Esto es debido a que cada sensor puede tener un offset diferente. El programa utilizado para la calibración del sensor se detalla en el Anexo 3.

Una vez el sensor se ha calibrado, está listo para recoger los datos. En este punto se ha redactado el código del programa para el cálculo de Co2 en ppm y que este sea transmitido hasta la base de datos. El programa utilizado se detalla en el Anexo 4.

Para este programa se han utilizado las siguientes librerías:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
```

La primera de ellas es una librería que cuenta con funciones que permiten la conexión de un módulo wifi de la familia ESP8266 a internet. Es una de las librerías más utilizadas para estos módulos y el programa implementado se ha utilizado para la conexión del módulo ESP8266 ESP-01 a la red. La segunda de ellas es una librería que cuenta con funciones que permiten utilizar el protocolo MQTT, por lo que, en el caso del programa utilizado en el proyecto, ha permitido conectarse al MQTT broker y mandar datos en un tema. También permite la recepción de datos entre muchas otras cosas que podrían ser utilizadas en el futuro del proyecto.

El programa en cuestión consta de las siguientes partes:

1. El módulo Wifi ESP8266 ESP-01 se conecta a un Wifi cercano obteniendo una IP.
2. Se configura el servidor MQTT con la URL y el puerto del mismo.
3. Se realiza la conexión al MQTT broker.
4. Se recoge el valor del Co2 en ppm.
5. Se espera hasta el próximo momento en el que se recogen los datos.

Las partes 3, 4 y 5 se repiten de forma infinita y si el módulo wifi se desconecta de la red se volverían a realizar todas las partes.

4.1.2 Comunicación

La parte de comunicación, como ya se ha comentado, es la encargada de transmitir los datos entre el componente hardware y la aplicación final .NET Core 5.

Esta parte está compuesta por un servidor donde se puede encontrar un Docker con dos contenedores. Cada uno de los contenedores alberga una aplicación necesaria para la comunicación entre el sistema.

A continuación, se van a describir en detalle cada uno de los componentes.

4.1.2.1 Servidor

Desde la empresa i3code han puesto a disposición de este trabajo una máquina virtual en Azure con Ubuntu 18.04.5 LTS. El resto de las aplicaciones han tenido que ser instaladas con posterioridad. En el apartado 3 del Estado del arte se ha concluido que la mejor forma para implementar tanto el MQTT broker como la aplicación de Node-red era hacerlo en contenedores. Una de las posibilidades que ofrecen los contenedores y que es muy atractiva para este proyecto es la posibilidad de compartir estos contenedores con la certeza de que donde se desplieguen van a funcionar de la misma forma. Para la creación de los contenedores, se ha instalado Docker mediante un script que ofrece la propia documentación de Docker como se puede ver en el Anexo 5. Una vez ya se dispone de Docker se pueden comenzar a crear los contenedores que se necesitan.

4.1.2.2 Contenedores

El primer contenedor creado ha sido el que integra el MQTT broker Mosquitto. La imagen de este Docker ya está diseñada y disponible en el Docker Hub [16] por lo que se ha creado una nueva imagen a partir de esta añadiendo las modificaciones necesarias para que funcione la comunicación entre el dispositivo IoT y la aplicación Mosquitto. Para la creación de esa nueva imagen, se han realizado unas primeras configuraciones con la imagen que se puede obtener de Docker Hub como se puede ver en el Anexo 6.

El segundo contenedor creado ha sido el que integra Node-red. La imagen de este Docker también está ya diseñada y disponible en el Docker Hub [17], por lo que como en el caso de Mosquitto, se ha creado una imagen a partir de la disponible en internet con las modificaciones necesarias para que funcione la comunicación entre la aplicación Mosquitto y Node-red. Para la creación de esa nueva imagen se han realizado también primeras configuraciones con la imagen disponible en Docker Hub como se puede ver en el Anexo 7.

Conexión entre contenedores

Los contenedores se pueden desplegar y asociar su ejecución a distintas redes, a ninguna red o *none network*, a la red en modo puente o *bridge network* y a la red en modo anfitrión o *host network*.

En la configuración *none network*, los contenedores no están conectados a ninguna red y por tanto se ejecutan en una red aislada sin poder ser accesibles desde fuera del contenedor. Estos contenedores tampoco pueden acceder a otros contenedores que se encuentren en diferentes redes. La red en modo puente o *bridge network*, es la red por defecto a la que se conectan los contenedores siendo esa una red interna y privada que crea el Docker Host para que se conecten sus contenedores. Cada contenedor obtiene una IP dentro de esta red, normalmente con el rango 172.17.0.0, por lo que los contenedores se pueden comunicar entre si a través de estas IP internas, pero no son accesibles desde fuera de esta red. Para ello se debe mapear los puertos del contenedor con puertos del Docker Host, para conseguir que todo lo que llegue al Docker por un puerto especifico llegue hasta el contenedor. De esta forma, dos contenedores que utilicen el mismo servicio y por tanto el mismo puerto pueden estar ejecutándose al mismo tiempo siempre que se mapeen o se enlacen con diferentes puertos en el Docker Host. La red anfitriona o *host network*, es la red del Docker Host, la única red accesible desde fuera del Docker. Al asociar la ejecución de un contenedor a ella, este será accesible desde fuera sin la necesidad de mapear puertos, ya que el puerto del Docker Host actuará como si fuera el propio puerto del contenedor para el servicio que ejecuta. De esta manera, dos contenedores que ejecuten el mismo servicio no podrán estar configurados de esta forma ya que un mismo puerto del Docker Host no puede actuar como los dos puertos de los contenedores.

Las ejecuciones de contendores de este proyecto se han asociado a una red en modo puente y como se puede ver tanto en el Anexo 6, como en el Anexo 7, se han mapeados los puertos de estos contenedores a los puertos del Docker Host. De esta forma el componente hardware puede acceder a el MQTT broker dentro del contenedor de Mosquitto a través de la red del Docker Host. En el caso de la comunicación entre el contenedor de Mosquitto y el de Node-red está la opción de que se comuniquen a través de las IP públicas dentro de la red puente o a través de la red del Docker Host, pero se ha decidido hacer a través de las IP públicas como se verá posteriormente.

Contenedor Mosquitto

Para el contenedor de Mosquitto pocos cambios han sido necesarios. En el Anexo 6 se describe la instalación de la imagen de Mosquitto y como se ha configurado la misma. Esta ha sido la única configuración necesaria ya que el MQTT broker Mosquitto simplemente debe recibir los mensajes publicados por un publisher y enviárselos a los suscriptores de ese tema por lo que, al menos en este trabajo, solo ha hecho falta la configuración inicial.

Contenedor Node-red

En el contenedor de Node-red se han programado los flujos necesarios. Como primera prueba se realizó un flujo simple que recogiera los datos del MQTT broker en el mismo tema que publicaba el Arduino los datos recogidos por el sensor y los escribiera en un documento dentro del servidor.

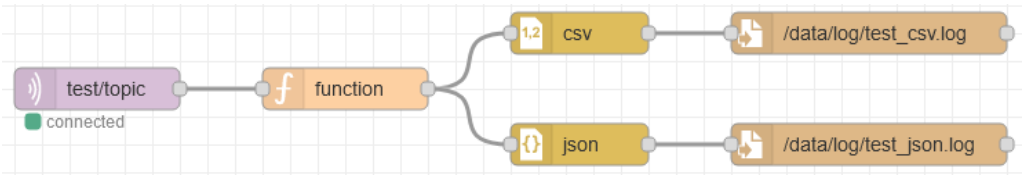

Ilustración 11: Fujo simple en Node-red

Con este flujo se conseguía escribir en los ficheros test_csv.log y test_json.log los datos recibidos en el nodo "test/topic". En estos ficheros se escribían los datos en modo CSV o JSON gracias al nodo de la función y de los nodos de los ficheros.

El nodo que recoge los datos de Node-red simplemente se configura añadiendo la IP del MQTT broker, que es la 127.17.0.2 y el puerto que es el 1883, el puerto común para las comunicaciones a través de MQTT, el tema de la comunicación y el tipo de QoS que se desea.

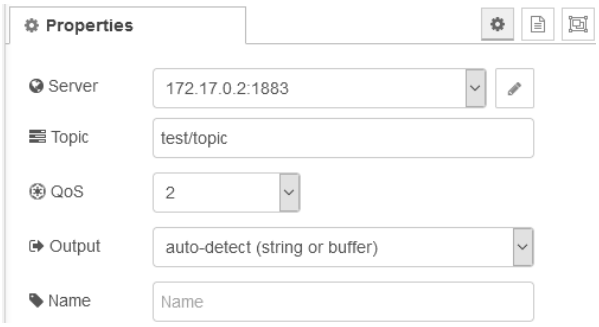

Ilustración 12: Configuración del nodo MQTT del flujo simple

El código del nodo de la función es el siguiente, en él se recoge la fecha y se crea un JSON con esta y el payload del mensaje recibido por el nodo "test/topic", que en cuestión es el mensaje enviado por el Arduino.

```
var d = new Date();
var t = d.getTime();
payload = {"time": d, "payload": msg.payload};
msg.payload = payload;
return msg;
```

Los nodos "csv" y "json" simplemente recogen el mensaje creado en la función y lo convierten al tipo correspondiente. En el caso del nodo "csv" se escoge la forma en la que separar los datos, teniendo la opción de hacerlo por comas, tabulaciones, espacios, etc.

La siguiente prueba ha sido la de enviar los datos recibidos hasta la API que se encuentra en el servidor, para ello el flujo ha sido el siguiente:

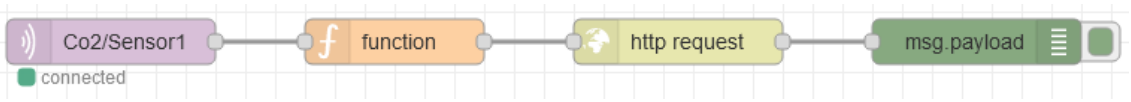

Ilustración 13: Flujo Node-red desde MQTT broker hasta API

Se puede apreciar que se cuenta con el mismo nodo MQTT, con un nodo función que se comentará a continuación y en este caso aparece un nodo “http request” y un nodo que permite ver el mensaje que devuelve la petición HTTP en una pantalla dentro de Node-red.

El nodo de MQTT está configurado de la misma forma al anterior, ya que la IP del servidor MQTT es la misma. El tema es algo diferente ya que indica tanto el tipo de dato que envía el sensor como su nombre. Se ha decidido que cada sensor cuente con un tema único, pero para evitar duplicar el flujo lo recomendable seria tener un único tema y que cada sensor enviara no solo el dato recabado, sino también el tipo de datos y su nombre o identificador.

El código del nodo “function” es algo diferente, ya que, para que el API REST guarde los datos en la base de datos necesita los parámetros del valor de Co2, la clave que equivale al tipo de dato que se va a enviar y el identificador del sensor que ha recogido los datos. Como se ha comentado, cada sensor cuenta con un tema diferente, por lo que en este nodo se puede añadir de forma manual el tipo de dato, así como el identificador del sensor. En el caso del valor de Co2 se añade el payload del mensaje recibido. En resumen, esta función crea un objeto llamado msg, en sus atributos añade los parámetros necesarios por el API (clave, valor y sensorId) y devuelve este objeto al siguiente nodo.

```
msg.clave = "Co2";
msg.valor = msg.payload;
msg.sensorId = 4;
return msg;
```

El nodo “http request” se encarga de realizar la petición HTTP como su nombre indica, en este caso se trata de una petición POST, pero se puede hacer cada una de las peticiones CRUD (GET, POST, PUT y DELETE) así como alguna otra. En el caso de realizar una petición POST se debe indicar la URL de la petición, si se desea activar la conexión segura, si se necesita autenticación y un par de configuraciones más, así como el tipo de respuesta que se recibe y el nombre del nodo, si se desea dar un nombre característico.

En cuanto a la URL, se pueden añadir los parámetros a enviar y en el caso actual, como se recoge un objeto desde el nodo anterior se puede hacer referencia al mismo mediante corchetes de la siguiente forma:

```
https://i3codemcampos.azurewebsites.net/ApiREST/AddTelemetry?clave={{{clave}}}&valor=
{{{valor}}}&sensorId={{{sensorId}}}
```

En este caso, se recogen los atributos del objeto `msg` y se envían en la petición a través de los argumentos de la forma `{{nombre_atributo}}`

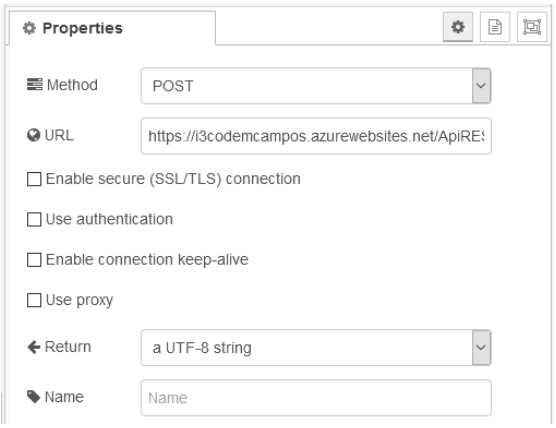

Ilustración 14: Configuración del nodo MQTT del segundo flujo

El último nodo que se encuentra en este flujo, es un nodo debug que permite depurar. En el caso actual se utiliza para recoger el mensaje devuelto por la petición HTTP y mostrar su payload, que se trata del mensaje devuelto por la función del API REST. Esta función, a modo de depuración también, devuelve una cadena en la que detalla los datos recibidos de la forma:

```
“Clave:” + clave + “ Valor:” + valor + “ sensorId:” + sensorId;
```

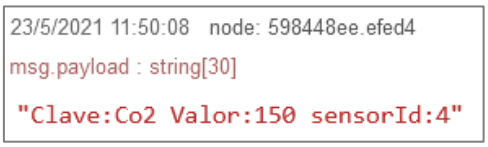

Ilustración 15: Mensaje de depuración obtenido del nodo debug

A la hora de desplegar la aplicación y utilizar el sistema en producción y no en pruebas, este nodo debería ser sustituido por un nodo que escriba en un fichero registro, de tipo log, los datos enviados por el sensor, quedando el flujo de la siguiente forma.

Ilustración 16: Flujo de Node-red para producción

La función del API devolvería la siguiente cadena:

```
fecha + “Clave:” + clave + “ Valor:” + valor + “ sensorId:” + sensorId;
```

Y en el fichero “datos_co2.log” se guardarían los datos de la siguiente forma:

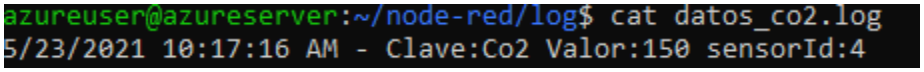

Ilustración 17: Fichero log para guardar los datos en producción

4.2 Implementación software

Para la implementación de la parte software, se ha tenido en cuenta como la empresa i3Code Solutions implementa sus proyectos. Debido a que el fin de este proyecto es conseguir una integración entre dispositivos IoT y las implementaciones software de las que disponen los clientes de i3Code Solutions.

Para la realización de esta parte se ha escogido un modelo de desarrollo en cascada o secuencial. En este modelo las etapas están muy definidas y se ejecutan una sola vez, además, cada etapa solo comienza cuando la anterior se da por finalizada.

Existen otros modelos de desarrollo que se denominan Ágiles. Estos modelos se basan, a grandes rasgos, en el trabajo conjunto con el cliente ya que se mantienen muchas reuniones con ellos y se les muestran muchas versiones para poder definir concretamente que es lo que quieren. Los proyectos en este tipo de modelos de desarrollo están en constante evolución, cambiando con los nuevos requisitos o modificaciones que el cliente propone.

En el caso de este proyecto, no se cuenta con la figura de un cliente, si no que los requisitos se han obtenido con el estudio de los datos que puede ofrecer el sistema hardware, por tanto, una metodología ágil no es lo más adecuada para el proyecto.

4.2.1 Etapas

Las etapas dentro del desarrollo en cascada son fundamentales y, como ya se ha comentado, están muy definidas. Al finalizar cada una de ellas se obtiene un resultado que se utiliza para comenzar la siguiente. A continuación, se van a definir estas etapas.

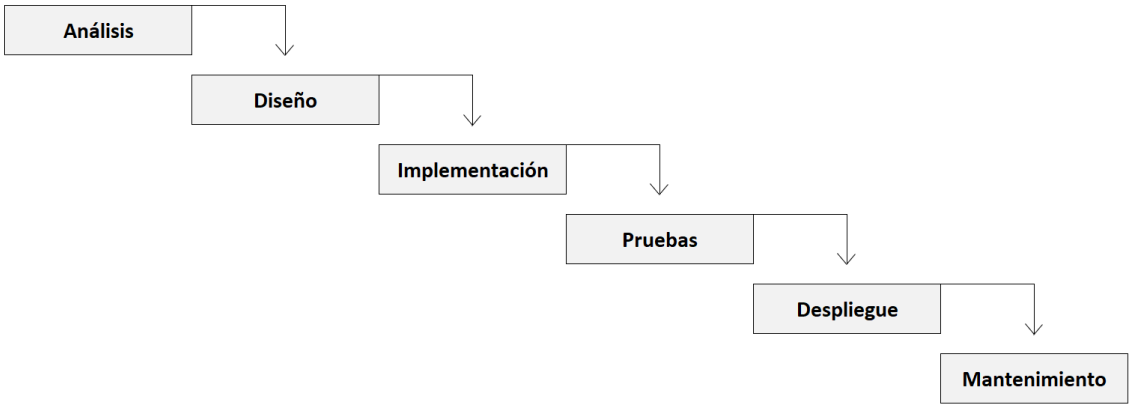

Ilustración 18: Etapas del desarrollo en cascada

Análisis

El software es siempre parte de un sistema mayor, por tanto, la primera fase es siempre el análisis de ese sistema para establecer los requisitos del sistema completo. Una vez conocidos los requisitos se han de dividir entre requisitos funcionales y no funcionales, además de analizarlos para comprenderlos por completo. Esta fase da como resultado un conjunto de requisitos a desarrollar.

Diseño

Después del análisis, esta segunda fase se encarga de conseguir el diseño de arquitectura del software, así como un plan de diseño detallado. El diseño de alto nivel se centra en la estructura de la solución identificando las partes principales y sus relaciones definiendo la arquitectura, en cambio, el diseño detallado se centra en los algoritmos empleados y la

organización del código para empezar la implementación. Esta fase da como resultado un borrador de producto final a alto y bajo nivel.

Implementación

Tras el diseño, se procede a la implementación de la arquitectura definida. Se implementa haciendo uso de las estructuras de datos diseñadas en la fase anterior. En esta fase se implementan los componentes por separado, se comprueban mediante las pruebas unitarias y se integran de forma gradual. En la fase de implementación es muy común la reutilización de código de otros proyectos, además de la utilización de bibliotecas, librerías o incluso la creación de estas últimas para poder utilizarlas en proyectos futuros. También es frecuente, en caso de estar desarrollando para un cliente, crear pequeñas versiones del producto para recibir validaciones y la opinión del mismo. Esta fase da como resultado el producto final.

Pruebas

Una vez se ha implementado la solución y se dispone del producto final, no se puede dar por concluido el proyecto, es necesaria una fase de pruebas. En esta fase se pretende validar el código, así como el funcionamiento del producto. Los errores encontrados son resueltos y se vuelve a probar hasta que no se encuentran más errores. En esta fase puede participar también el cliente probando de la forma más realista posible la solución para validarla por su parte. Esta fase da como resultado el producto final validado.

Despliegue

En el momento en el que ya se dispone del producto final validado se puede desplegar en el entorno de producción, este es el entorno en el que va a ser utilizado. Tras probar que todo funciona correctamente se puede concluir que el producto se ha desplegado. Esta fase da como resultado el producto corriendo en el entorno de producción y por tanto entregado al cliente.

Mantenimiento

A partir del despliegue se deben destinar recursos al mantenimiento, este mantenimiento consiste en modificaciones del producto que abarcan los propios fallos que puedan aparecer en producción por el uso de los clientes, cambios en el entorno de producción que requieran de una actualización del producto, cambios que pueden aumentar el rendimiento del producto o ampliaciones de requisitos y funcionalidades que desee el cliente. Esta fase suele dar como resultado un contrato con el cliente sobre cómo actuar ante posibles actualizaciones.

4.2.2 Análisis

En la etapa de análisis se ha comentado que se obtiene una lista de requisitos a desarrollar, estos requisitos se dividen entre los requisitos funcionales y los requisitos no funcionales. Los requisitos que especifican los aspectos funcionales del software son los requisitos funcionales, definen las funcionalidades proporcionadas por los sistemas o componentes. Por el contrario, los requisitos que no están relacionados con el aspecto funcional del software son los requisitos no funcionales, definen las características esperadas de un software, usualmente son características que esperan los usuarios.

Los requisitos de este proyecto se han obtenido analizando las posibilidades que ofrecen los datos recabados, así como aquellas cosas que el usuario puede querer realizar. Esos datos son mayoritariamente la cantidad de Co2 de una habitación. Los requisitos, tanto funcionales como no funcionales, se detallan en el Anexo 8.

En esta etapa del análisis también se describen los casos de uso de la aplicación, estos son diagramas o tablas que pretenden documentar los requisitos del sistema indicando el comportamiento del mismo. En otras palabras, un caso de uso es una secuencia de interacciones entre el sistema y sus actores, en respuesta a un evento que inicia un actor principal (comúnmente el usuario). En ellos se especifica la comunicación y el comportamiento del sistema con los usuarios e incluso con otros sistemas. Los casos de uso se detallan en el Anexo 9

4.2.3 Diseño

En la etapa de diseño se ha comentado que se obtiene un borrador del producto final. Para realizar este borrador se ha utilizado la aplicación *Evolus Pencil* en la versión 3.1.0 [18] y se ha diseñado cada una de las páginas de la aplicación. Se pueden ver los diseños en el Anexo 10. En esta etapa también se ha diseñado la base de datos con sus tablas y los campos de cada una de ellas, en el Anexo 11 se puede ver tanto el modelo entidad-relación como el modelo relacional de la base de datos. El modelo entidad-relación es un modelo conceptual que representa los objetos y la relación entre ellos, en cambio, el modelo relacional representa las tablas, con sus atributos y las relaciones entre ellas. En el Anexo 11 se detallan más las características de estos modelos.

Durante este proceso, por lo general, se trabaja en conjunto con el cliente o al menos se tiene muy en cuenta su opinión tratando de que verifique la parte de más alto nivel, es decir, las páginas de la aplicación. Pero como ya se ha comentado, en este proyecto no se cuenta con un cliente como tal, por lo que esta parte del diseño ha sido marcada por ciertas pautas recibidas desde la empresa i3Code Solutions, así como las oportunidades que daban los datos que el sistema hardware iba a guardar.

4.2.4 Implementación

A la hora de la implementación, ha sido la empresa i3Code Solutions la que ha marcado los requisitos ya que este proyecto tiene como fin poder incorporar en las soluciones que tienen sus clientes una comunicación con un sistema de IoT. Por tanto, el lenguaje de programación utilizado ha sido *C#* se trata de un conjunto de lenguajes como *C*, *C++*, *Java* o *Visual Basic* creado por Microsoft. Recoge lo mejor de cada lenguaje, como la potencia de *C* o la facilidad de programación y aprendizaje de *Visual Basic*. Está orientado a objetos para toda la plataforma *.NET* de Microsoft. Más concretamente se ha utilizado la sintaxis *Razor Pages* que se basa en *C#,* aunque también admite *Visual Basic*. Esta sintaxis permite programar la parte de backend en *C#* y la parte de frontend en *HTML*, CSS o *JS.*

El IDE de desarrollo utilizado ha sido *Microsoft Visual Studio* compatible con lenguajes de programación como *C++*, *C#* o *Visual Basic* y disponible para los sistemas operativos *Windows* y *MacOS*. Además, permite realizar aplicaciones de escritorio, aplicaciones móviles, aplicaciones web *ASP .NET* y servicios web *XML*. El framework o entorno de trabajo utilizado ha sido *ASP .NET Core 5 MVC* que implementa el patrón modelo-vista-controlador, que separa los datos de aplicación, la interfaz de usuario y la lógica de control en tres componentes diferenciados, permitiendo crear aplicaciones o servicios web *.NET* y *C#* ampliando las bibliotecas y herramientas que componen el framework *.NET*. Para la conexión entre la aplicación web y la base de datos, la empresa i3Code suele utilizar el framework *Entity Framework* como *ORM* de la aplicación, ya que de manera muy sencilla convierte las estructuras en clases con las que poder trabajar usando objetos. La base de datos se ha gestionado e implementado utilizando el IDE *Microsoft SQL Management Studio*, que combina un amplio grupo de herramientas gráficas y editores de script que facilitan la gestión de las bases de datos.

4.2.5 Pruebas

En esta fase, como su nombre indica, se deben realizar pruebas del producto final. Se pretende obtener la solución validada ya sea por los propios desarrolladores haciendo pruebas o por el cliente tratando de utilizar la solución de la forma más completa posible y como se va a utilizar una vez esté desplegada en el entorno de producción.

En cuanto a esta fase, se han ido realizando pequeñas pruebas a medida que se iba desarrollando la solución para comprobar que funcionaba correctamente. En estas pruebas fueron apareciendo alguna serie de errores y se resolvieron al instante. Una vez terminada la solución completa, se han realizado las siguientes pruebas:

1. Una prueba por parte de la desarrolladora en la que se han probado todas las funcionalidades al completo y con todos los tipos de usuarios. En esta prueba ha aparecido algún pequeño error, pero una vez solucionado la aplicación ya estaba validada a ojos de su creadora.
2. Una prueba por parte de la desarrolladora y un voluntario tratando de ponerse en la piel de un usuario y realizando las acciones que este realizaría. El voluntario no conocía de nada la aplicación y se ha puesto a navegar encontrando algún pequeño error. Se ha considerado importante hacer este "experimento" ya no solo para poder encontrar nuevos errores sino también para conocer la usabilidad de la solución y si es intuitiva a ojos de una persona inexperta y que no la conocía previamente. Una vez solucionados los errores encontrados, a ojos de un usuario la solución ya estaba validada.

Tras realizarse ambas pruebas se ha considerado que se tenía una solución validada y lista para pasar a lo que sería el entorno de producción.

4.2.6 Despliegue

En la fase de despliegue se entrega el producto al cliente desplegándolo en el entorno de producción. En el caso actual ya se ha comentado en varias ocasiones que no se dispone de un cliente, por lo que esta fase ha sido algo distinta. En ella simplemente se ha desplegado el producto final en una serie de recursos en Azure para que ya pueda ser accedida desde cualquier navegador. Estos recursos con SQL server, SQL Database y APP Service.

SQL Server

Azure SQL Server o servidor SQL es un gestor de base de datos que permite, como su nombre indica, gestionar bases de datos relacionales. Se trata del mismo servicio que ofrece el IDE *Microsoft SQL Server Management,* pero trasladado a la web, por lo que se puede utilizar el mismo IDE que en el desarrollo para gestionar las bases de datos de producción.

SQL Database

Azure SQL Database o Base de datos de Azure, es un motor de base de datos de tipo PaaS. Esta base de datos se crea en Azure SQL Server por lo que es idéntica a la creada en el entorno de desarrollo en *Microsoft SQL Server Management*, por lo que solo ha sido necesario migrar la base de datos sin necesidad de crearla de nuevo. Además, este motor de base de datos es escalable y permite de forma fácil obtener más recursos a medida que la base de datos crece, algo muy interesante para soluciones en expansión o por ejemplo para el proyecto actual que ha empezado siendo un pequeño producto, pero si se comercializa puede crecer y esta funcionalidad sería muy útil.

APP Service

App Service o servicio de aplicaciones es un servicio que ya se ha estudiado con anterioridad en el Estado del arte. Se trata de un servicio que ofrece la posibilidad de crear, implementar y escalar aplicaciones web. Asegura ser una plataforma escalable de forma sencilla por lo que es una solución muy útil para este proyecto, ya que, si en un futuro se comercializa con esta solución, la propia plataforma permite crear una aplicación mayor y más potente sin necesidad de migrarla a otro servicio o a un servidor externo.

4.2.7 Mantenimiento

En la etapa del mantenimiento suele dar como resultado un contrato con el cliente en el que se acuerda el tipo de soporte que se va a dar sobre el producto entregado. En este caso ya se conoce que no hay cliente por lo que esta fase es difícil llevarla a cabo. Si en un futuro la empresa i3Code Solutions comercializa este producto podría llegar a haber acuerdos de mantenimiento o soporte con clientes.

5. Líneas futuras

Como se comentará más adelante, se han conseguido cumplir los objetivos, pero se puede considerar este un proyecto en evolución, ya que tanto la parte de red como la implementación software pueden evolucionar todavía mucho. Pensando en el posible futuro de este proyecto y en que puede llegar a ser comercializado por la empresa i3Code Solutions, se ha estudiado qué se podría añadir para hacer de este proyecto una mejor solución y se ha llegado a algún desarrollo concreto.

Uno de estos desarrollos ha sido la idea de añadir leds indicadores del nivel de Co2 al componente hardware del sistema, de esta forma se puede ver sin necesidad de acceder a la aplicación web, cual es el nivel de Co2 en la sala y por tanto cómo es la calidad del aire actual. Este desarrollo se ha llevado a cabo en forma de prueba y se han utilizado tres luces leds con los colores que aparecen en la página web, es decir, verde si la calidad del aire es buena, naranja o amarillo si la calidad empieza a ser discutible y rojo si la calidad es mala y se debe ventilar con urgencia. Para este desarrollo se han tenido que utilizar otros componentes hardware ya que el módulo wifi utilizado hasta ahora, como se puede ver en el Anexo 1, solo cuenta con dos pines de entrada y salida, y uno está ocupado por el sensor, por lo que quedaba un único pin para tres leds. El nuevo módulo wifi utilizado ha sido el descartado en el Estado del arte que como bien se comentaba se está utilizando en el momento que se ha detectado la necesidad de usar más pines. Este nuevo módulo wifi es el ESP8266 ESP-12, su montaje se puede ver en el Anexo 13 y el programa ejemplo cargado dentro del módulo ESP-12 se puede ver en el Anexo 14.

El programa está en proceso de pruebas todavía, pero se ha comprobado que en función de los valores recibidos por el sensor se enciende el led correspondiente. En el código del Anexo 14, el componente hardware recoge cada diez minutos el valor del Co2 del espacio donde se encuentra, si este valor supera unos límites marcados en el código, se enciende el led correspondiente durante los diez minutos de espera hasta la siguiente medición. En la primera versión, el led permanecía encendido durante dos segundos, pero se consideró que era un tiempo muy reducido y que era mejor idea que el sensor estuviera encendido hasta la próxima medición para que se pudiera ver con facilidad y no hubiera que estar atento a él. Como se acaba de comentar, los valores que marcan cuándo se debe encender un led, están descritos en el código como variables constantes, pero sería una mejor implementación si estos valores pudieran ser modificados desde la aplicación web por el propietario del sensor, como ya se hacen las alarmas de la aplicación. Por esto se ha comentado que este desarrollo se ha llevado a cabo en forma de prueba, porque se puede integrar de una manera más completa con el sistema. Por otro lado, el tiempo de espera entre medición y medición también es una constante dentro del programa del módulo, pero como en el caso de los valores que marcan el encendido de los leds, podría ser un tiempo modificable desde la aplicación web.

Otro de los desarrollos que se pueden añadir al proyecto es integrar un sensor de temperatura y humedad al sistema para así recoger valores más exactos de Co2 ya que la calidad del aire es sensible también a la temperatura y la humedad. En este caso se necesitaría usar también el módulo wifi ESP-12 ya que se necesita como mínimo otro pin de entrada y salida. En este caso se debería volver a calibrar el sensor de Co2 teniendo en cuenta el valor de la temperatura, pero el programa sería muy parecido al del Anexo 4 cambiando solo la forma de calcular el Co2 en ppm. A este montaje se le podrían añadir también los leds para que fuera lo más completo posible. Este desarrollo no se ha implementado porque no se disponía de un sensor de temperatura y humedad, pero si la empresa i3Code quisiera añadir este desarrollo al

trabajo se conocen cuáles son los pasos a seguir ya que es muy parecido a lo realizado anteriormente.

Dejando a un lado los desarrollos que completan la parte hardware, un posible desarrollo que sería importante tener en cuenta si se desea comercializar la aplicación web es comprobar su movilidad. El desarrollo de la aplicación web se ha basado en el uso a través del ordenador y en que en este tipo de dispositivos se vea bien y se puedan aprovechar todas las funcionalidades, en cambio, en dispositivos de tipo teléfonos móviles o tabletas es un trabajo que queda por hacer. Se ha comentado que sería importante a la hora de comercializar el producto, porque actualmente se utiliza el teléfono móvil para todo y es mucho más probable que se utilice la aplicación desde este tipo de dispositivos. En el caso de comercializarse solo para utilizar en ordenadores es una aplicación funcional y muy completa.

El último desarrollo que se va a comentar está relacionado con los contenedores. En este caso se vuelve a imaginar el caso de tener el sistema completo comercializado o implantado en la empresa de un cliente. En el caso de estar implementado en la empresa de un cliente, este ha colocado múltiples sensores en su empresa, estos envían datos cada tres minutos porque desde la empresa se ha considerado que mediciones cada diez minutos es mucho tiempo. Por tanto, el broker MQTT como Node-red y la API RESTful se encuentran con un gran número de datos en poco tiempo por lo que los contenedores podrían saturarse. En el caso de estar comercializado, se tendría una única aplicación web con varias empresas, dentro de ellas los usuarios y después vendrían los sensores. Se vuelve a tener múltiples datos de entrada a la API, así como en este caso a la aplicación web. Al tenerla en Azure, esta parte software del proyecto no correría mucho riesgo porque es escalable y simplemente se aumentarían los recursos disponibles, pero ¿qué ocurre entonces con los contenedores? estos podrían verse saturados como en el caso anterior.

La solución a lo anterior es Kubernetes, una orquestación que se ha estudiado en el apartado del Estado del arte y para la que Azure ofrece un servicio, así que no sería muy complejo de utilizar. Con Kubernetes se podrían crear varias instancias de los contenedores, para reducir el tráfico de estos y poder repartir el trabajo. Además, si un contenedor falla, se puede acceder a sus servicios a través del resto y Kubernetes se encarga de volverlo a poner en marcha. Además, Kubernetes es capaz de escalar el número de instancias según el uso de los usuarios, algo que sería muy interesante para pagar solo por el tráfico real de los datos y tener siempre el número mínimo de instancias, pero el tráfico bien repartido entre ellas. El servicio Kubernetes AKS de Azure ofrece la integración y distribución continuas (CD/CI). La integración continua (CI) tiene como objetivo establecer una forma coherente y automatizada para crear y probar aplicaciones. La distribución continua (CD) tiene como objetivo automatizar la entrega de aplicaciones a determinados entornos de infraestructura como puede ser el entorno de producción. En el caso del proyecto actual en los supuestos casos futuros, la integración continua podría ser muy útil para poder probar la aplicación web de forma automática y poder dedicar más recursos al desarrollo, por ejemplo, de lo comentado anteriormente de la movilidad en dispositivos móviles. La distribución continua es muy interesante a la hora de sacar nuevas actualizaciones y poder hacerlo de forma automática es una carga menos de trabajo, además en el trabajo automatizado no se va a olvidar de ninguna parte importante para el despliegue de la aplicación en cualquiera de los entornos e infraestructuras.

Estas son solo una parte de todos los posibles desarrollos dentro de este proyecto en evolución y que harían de él un proyecto más completo y preparado para desplegarse en empresas o incluso comercializarse.

6. Conclusión

El presente proyecto tenía como objetivo integrar y programar dispositivos de IoT para detectar la calidad del aire en interiores, así como diseñar e implementar la comunicación entre estos dispositivos y una aplicación .NET Core 5 estudiando también las tecnologías y protocolos que pueden ser utilizadas en el desarrollo de dicha comunicación.

Se puede concluir que los objetivos del proyecto se han logrado con creces, se ha montado y programado un sistema IoT que mide la calidad del aire en cantidad de Co2 (ppm) mediante un Arduino, un módulo wifi y un sensor de Co2. También se ha diseñado e implementado la comunicación de este dispositivo y una aplicación web .NET Core 5 mediante los contenedores de Mosquitto, y Node-red dentro de un Docker y una API RESTful. Además, todas las tecnologías y protocolos han sido seleccionados tras su estudio dentro del apartado de Estado del arte.

Este proyecto ha tenido evolución durante el desarrollo desde el primer día ya que se tuvo que decidir desde qué hardware utilizas hasta dónde desplegar las tecnologías y aplicaciones a utilizar. Ha sido un proyecto con un objetivo muy claro pero abierto a muchas implementaciones diferentes y eso ha hecho que sea un proyecto más interesante ya que ha obligado a estudiar, conocer y tratar de decidir qué implementación era la mejor. Se ha tratado de escoger aquellos dispositivos, protocolos o servicios que hacían de este proyecto un proyecto más completo y cumplían con los objetivos iniciales. Esta elección se ha hecho tras el estudio y la realización de unas tablas, en las que se puntuaban las opciones a elegir teniendo en cuenta los requisitos del proyecto y la opción que mayor puntuación obtenía, siendo esta la escogida. Es muy probable que, con otro tipo de hardware y otra implementación diferente, el proyecto hubiera sido completamente diferente y hubiera obtenido otros resultados.

Durante el desarrollo también se han encontrado algunos problemas debido al desconocimiento del uso y funcionamiento de muchas de las tecnologías utilizadas, pero tras la formación en dichas tecnologías los problemas se han ido solucionando poco a poco hasta poder llegar a tener un proyecto completamente funcional. Algunos de los problemas se dieron con el despliegue de los contenedores en Docker y sus configuraciones o con la programación en forma de nodos en Node-red. Otro de los problemas llegó con la forma de utilizar o contratar los servicios de Azure, pero con la documentación que ofrece Microsoft se han podido solucionar.

El proyecto ha supuesto un gran reto tanto personal como profesional desde el primer día para mí. Ha sido la primera vez que he tenido que estudiar este tipo de tecnologías y protocolos por mi cuenta, así como decidir cuáles utilizar estando toda la responsabilidad del proyecto sobre mí. Ha sido una muy bonita experiencia poder crear la parte hardware con el Arduino y los sensores, programarlo y comprobar su funcionamiento individual, ha sido de las partes que más me ha gustado y con las que más he aprendido junto con la implementación de la comunicación con la que he aprendido a formarme, buscar y decidir por mi cuenta. La parte de desarrollo de la aplicación web tal vez haya sido la más sencilla, pero con ella he cerrado un ciclo de comunicación mostrando los datos recabados por el sensor que puede encontrarse en una localización completamente diferente.

Como ya he comentado en el apartado de las Líneas futuras estoy más que dispuesta a seguir trabajando en el proyecto si la empresa i3Code apuesta por él. Me encantaría poder trabajar en los posibles desarrollos comentados anteriormente o en otros diferentes para hacer de este proyecto un producto propio de i3Code comercializable. Aparte de esto, me parecería muy interesante seguir con este proyecto ya que es un proyecto que pretende ayudar a la

calidad de vida de las personas o más en concreto de los trabajadores de las empresas en varios sentidos. El primero de ellos es por el uso que se le puede dar a este proyecto completo, recogiendo la calidad del aire y avisando de cuándo es preciso ventilar una habitación porque el aire no tiene la calidad mínima necesaria. Algo que por el actual estado de la pandemia es muy interesante, pero que también se le debería prestar atención cuando esto pase por la propia salud de nuestro cuerpo. Otro de los motivos por los que este proyecto puede ayudar a la calidad de vida de las personas es, ya no tanto en cuanto a la salud, si no por la posibilidad de integrar cualquier dispositivo IoT con conectividad a internet en un desarrollo software de .NET Core 5. Ya que la integración del dispositivo de este proyecto con la aplicación web de este se podría reutilizar en otros proyectos para conectar otros dispositivos a otra aplicación web ya creada.

En conclusión, este proyecto ha cumplido con sus objetivos y podría seguir mejorándose e implementando actualizaciones en varios aspectos. Aún sin ellos, se trata de un proyecto completamente funcional que recorre todo el ciclo de una comunicación desde un dispositivo IoT hasta una visualización de datos final en una aplicación web, tratándose así de un proyecto listo para ser comercializado o implantado en cualquier empresa como sistema de calidad de aire.

7. Bibliografía y referencias

[1] "Tutorial sensores de gas MQ2, MQ3, MQ7 y MQ135", *Naylamp Mechatronics*, 2016. [En línea]. Disponible en https://naylampmechatronics.com/blog/42_tutorial-sensores-de-gas-mq2-mq3-mq7-y-mq135.html. [Accedido: 24-feb-2021]

[2] Hita, A. "MQYY vs HTTP: ¿Qué protocolo es mejor para IoT?", BBits Tecnología y Opinión, 2020. [En línea]. Disponible en https://borrowbits.com/2020/04/mqtt-vs-http-que-protocolo-es-mejor-para-iot/. [Accedido: 18-mar-2021]

[3] Bartnitsky, J. "HTTP vs MQTT performance tests", Flespi, 2018. [En línea]. Disponible en https://flespi.com/blog/http-vs-mqtt-performance-tests. [Accedido: 18-mar-2021]

[4] "¿Qué es MQTT? Su importancia como protocolo IoT", Luis Llamas, 2019. [En línea]. Disponible en https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/. [Accedido: 22-feb-2021]

[5] Wähner, K. "IoT Open Source Integration Comparison (Kura, Node-RED, Flogo, Apache Nifi, StreamSets)", Slideshare, 2016. [En línea]. Disponible en https://es.slideshare.net/KaiWaehner/iot-open-source-integration-comparison-kura-nodered-flogo-apache-nifi-streamsets. [Accedido: 08-mar-2021]

[6] "Gateways IoT", Aprende Arduino, 2018. [En línea]. Disponible en https://aprendiendoarduino.wordpress.com/2018/11/21/gateways-iot/. [Accedido: 08-mar-2021]

[7] "Eclipse Mosquitto™ An open source MQTT broker". Mosquitto.org, s.f. [En línea]. Disponible en https://mosquitto.org/. [Accedido: 22-mar-2021]

[8] "Docker: qué es y porque es tan popular", Soltel Group, 2018. [En línea]. Disponible en https://www.soltel.es/docker-que-es-y-por-que-es-tan-popular/. [Accedido: 05-mar-2021]

[9] "Build and Ship any Application Anywhere", Docker Hub, s.f. [En línea]. Disponible en https://hub.docker.com/. [Accedido: 12-mar-2021]

[10] "¿Qué es la arquitectura orientada a los servicios (SOA)?", Red Hat, 2020. [En línea]. Disponible en https://www.redhat.com/es/topics/cloud-native-apps/what-is-service-oriented-architecture [Accedido: 22-mar-2021]

[11] Sandy. "Conceptos básicos de Servicios Web SOAP, WSDL y XSD", Desarrollo con SOA, 2010. [En línea]. Disponible en http://desarrolloconsoa.blogspot.com/2014/02/conceptos-basicos-de-servicios-web-soap.html. [Accedido: 22-mar-2021]

[12] Paszniuk, R. "Web Service Soap con Java EE". Programación.com.py, 2014. [En línea]. Disponible en https://www.programacion.com.py/web/java-web/web-service-soap-con-java-ee. [Accedido: 22-mar-2021]

[13] Fielding, R. "Architectural Styles and the Design of Network-based Software Architectures". University of California, Irvine, 2000. [En línea]. Disponible en https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. [Accedido: 22-mar-2021]

[14] "Virtual Machines". Azure, s.f. [En línea]. Disponible en https://azure.microsoft.com/es-es/services/virtual-machines/. [Accedido: 22-mar-2021]

[15] "Servicios de contenedor" Azure, s.f. [En línea]. Disponible en https://azure.microsoft.com/es-es/product-categories/containers/. [Accedido: 22-mar-2021]

[16] "eclipse-mosquitto", Docker Hub, 2021. [En línea]. Disponible en https://hub.docker.com/_/eclipse-mosquitto. [Accedido: 17-mar-2021]

[17] “nodered/node-red”, Docker Hub, 2021 [En línea]. Disponible en https://hub.docker.com/r/nodered/node-red. [Accedido: 23-mar-2021]
[18] “Pencil Project”. Evolus Pencil, 2019. [En línea]. Disponible en https://pencil.evolus.vn/ [Accedido: 9-mar-2021]
[19] “ASP.NET”. Microsoft, s.f. [En línea] Disponible en https://dotnet.microsoft.com/apps/aspnet. [Accedido: 20-abr-2021]
[20] Marco, F. “Diseño e implementación de un sistema de medida de gases con Arduino” Universidad de Zaragoza, 2016. [En línea]. Disponible es https://zaguan.unizar.es/record/59102/files/TAZ-TFG-2016-2689.pdf [Accedido: 21-abr-2021]
[21] “TECHNICAL DATA MQ-135 GAS SENSOR” Olimex [En línea]. Disponible en: https://www.olimex.com/Products/Components/Sensors/Gas/SNS-MQ135/resources/SNS-MQ135.pdf [Accedido: 21-abr-2021]
[22] “Install Docker Engine on Ubuntu”, Docker docs, 2021 [En línea]. Disponible en https://docs.docker.com/engine/install/ubuntu/. [Accedido: 12-mar-2021]

Anexos

Anexo 1 - Montaje del componente hardware

Para el montaje del componente hardware se ha tenido en cuenta que el módulo wifi dispone de dos modos, el modo de programación y el modo de ejecución. Para estos modos el montaje varía ligeramente. En ambos modos el Arduino debe tener desconectado su microcontrolador o conectado el pin RST (reset) con GND (ground) o tierra para que el Arduino solo funcione como una herramienta de conexión entre el módulo y el ordenador ya que este pequeño modulo no cuenta con una conexión directa integrada. La pequeña variación que existe entre los modos de funcionamiento se consigue conectando de forma diferente los pines del módulo wifi. Los pines que se encuentran en el módulo son los siguientes:

Ilustración 19: Pines del módulo wifi

Pines	Traducción	Función
3V3	3,3V	Pin que alimenta el módulo, debe estar conectado a 3,3V.
RST	Reset	Pin que realiza el reinicio del módulo.
EN / CH_PD	Enable / chip power-down	Pin que apaga o enciende el módulo según su conexión.
TX	Transmit	Pin que transmite la información del procesador.
RX	Receive	Pin que recibe información que es para el procesador.
IO 0 / GPIO 0 IO 2 / GPIO 2	General Purpose Input/Output	Pin de Entrada/Salida de propósito general, pin digital.
GND	Ground	Pin que debe ser conectado a la toma de tierra.

Tabla 11: Pines del módulo ESP8266 ESP-01

Para que el módulo wifi funcione, tanto su pin 3V3, como el pin EN / CH_PD deben estar conectados a una tensión de 3,3V y el pin GND debe estar conectado. Además, si se quiere que su procesador envíe y reciba datos deberán estar conectados también sus pines TX y RX. El resto de los pines serán los encargados de definir en qué modo se encuentra el módulo y de que este reciba datos externos.

Modo de programación

Para este modo no es necesaria la presencia del sensor de Co2, simplemente se necesita el módulo wifi.

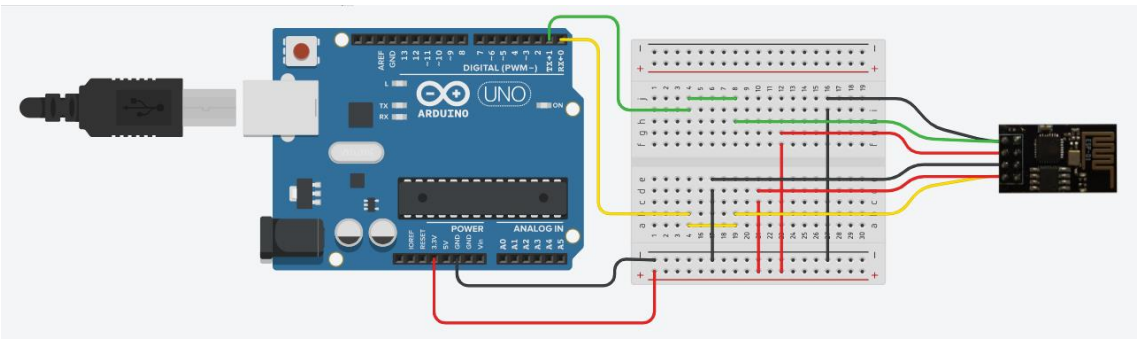

Ilustración 20: Montaje modo programación

Módulo wifi	Arduino
3V3	3,3V
RST	-
EN / CH_PD	3,3V
TX	TX
RX	RX
GPIO 0	GND
GPIO 2	-
GND	GND

Tabla 12: Conexiones modo programación

Para que el módulo wifi se encuentre en modo de programación, su pin GPIO 0 debe estar conectado a tierra, su pin RST desconectado y sus pines TX y RX conectados a los mismos pines del Arduino, lo que hará que la información que reciba el Arduino pase directamente al módulo y viceversa. El pin GPIO 2 pueden utilizarse como entrada o salida de datos, pero en este modo no es importante.

Modo de ejecución

Para este modo el montaje del módulo es muy parecido, pero en este caso es necesario añadir el sensor de Co2 para poder enviar los datos que recoja.

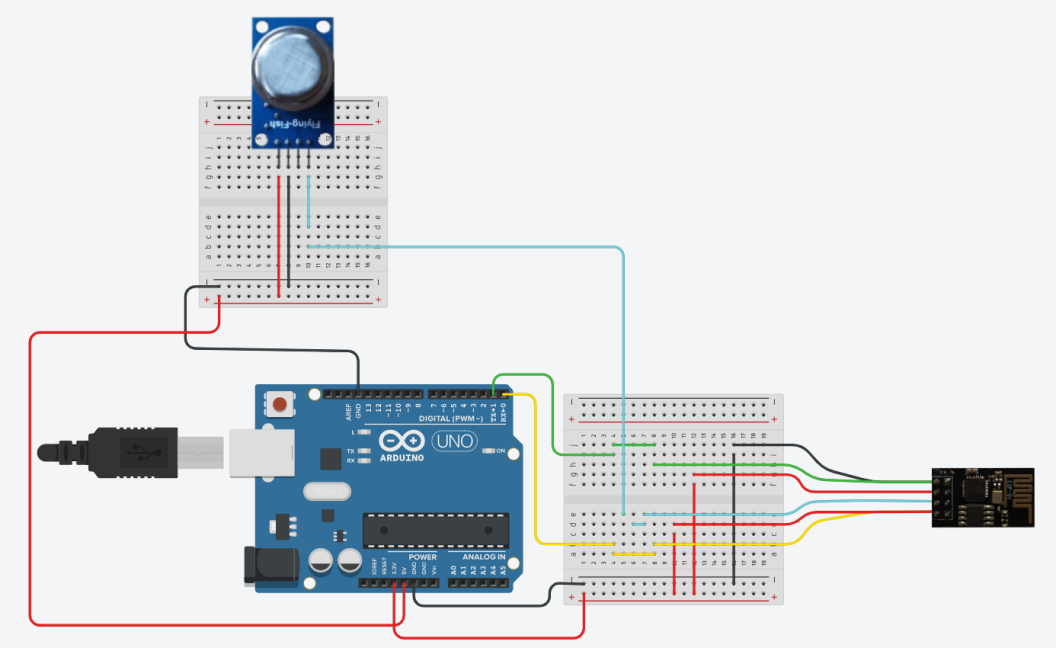

Ilustración 21: Montaje modo ejecución

Módulo wifi	Arduino
3V3	3,3V
RST	-
EN / CH_PD	3,3V
TX	TX
RX	RX
GPIO 0	-
GPIO 2	-
GND	GND

Tabla 13: Conexiones modo ejecución

Para que el módulo wifi se encuentre en modo de ejecución, su pin RST debe estar desconectado y sus pines TX y RX conectados a los mismos pines del Arduino, como en el modo anterior. En este caso los pines GPIO 0 y GPIO 2 pueden utilizarse como entrada o salida de datos, algo que en este modo sí que es importante para que el módulo pueda recibir información, por ejemplo, de un sensor, como es el caso de este proyecto.

Anexo 2 - Calibración del sensor MQ135

Para la calibración del sensor MQ135, tras recoger información sobre cómo proceder [20], se ha concluido que es necesario ajustar la curva de la medición de Co2 del sensor a una función exponencial de la forma $y = a * x^b$.

Para ello se ha obtenido de la gráfica de la ficha técnica del sensor [21].

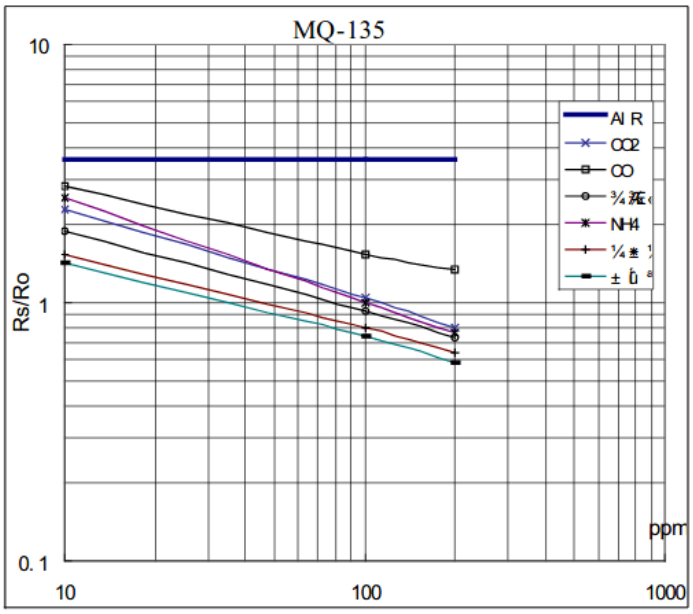

Ilustración 22: Características sensitivas del sensor MQ135

A continuación, se ha calculado la función exponencial y para ello se han recogido los siguientes datos de la gráfica.

Eje de abscisas	Eje de ordenadas
10	2,3
17	2
20	1,85
30	1,67
40	1,52
50	1,4
60	1,31
70	1,25
80	1,19
90	1,16
100	1,07
150	0,91
200	0,8

Tabla 14: Datos obtenidos de la gráfica del sensor MQ135

Se tienen las ecuaciones $y_i = a * x_i^b \ \ \forall i \in 1, \dots, 15$

A continuación, se han añadido logaritmos neperianos a ambos lados y obtenido un sistema matricial con el que poder sacar el valor de las variables.

$$\ln(y_i) = \ln(a * x_i^b)$$

$$\ln(y_i) = \ln(a) + \ln(x_i^b)$$

$$\ln(y_i) = \ln(a) + \ln(x_i) * b$$

El sistema matricial que ayuda a obtener las incógnitas es el siguiente:

$$\begin{pmatrix} 1 & \ln(10) \\ 1 & \ln(17) \\ 1 & \ln(20) \\ 1 & \ln(30) \\ 1 & \ln(40) \\ 1 & \ln(50) \\ 1 & \ln(60) \\ 1 & \ln(70) \\ 1 & \ln(80) \\ 1 & \ln(90) \\ 1 & \ln(100) \\ 1 & \ln(150) \\ 1 & \ln(200) \end{pmatrix} * \begin{pmatrix} \ln(a) \\ b \end{pmatrix} = \begin{pmatrix} 1 & \ln(2{,}3) \\ 1 & \ln(2) \\ 1 & \ln(1{,}85) \\ 1 & \ln(1{,}65) \\ 1 & \ln(1{,}52) \\ 1 & \ln(1{,}4) \\ 1 & \ln(1{,}31) \\ 1 & \ln(1{,}25) \\ 1 & \ln(1{,}19) \\ 1 & \ln(1{,}16) \\ 1 & \ln(1{,}07) \\ 1 & \ln(0{,}91) \\ 1 & \ln(0{,}8) \end{pmatrix}$$

Para resolver este sistema, se ha multiplicado cada parte de la igualdad por la traspuesta de la matriz de coeficientes para obtener así un sistema con una única solución.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \ln(10) & \ln(17) & \ln(20) & \ln(30) & \ln(40) & \ln(50) & \ln(50) & \ln(60) & \ln(70) & \ln(80) & \ln(90) & \ln(100) & \ln(150) & \ln(200) \end{pmatrix} * \begin{pmatrix} 1 & \ln(10) \\ 1 & \ln(17) \\ 1 & \ln(20) \\ 1 & \ln(30) \\ 1 & \ln(40) \\ 1 & \ln(50) \\ 1 & \ln(60) \\ 1 & \ln(70) \\ 1 & \ln(80) \\ 1 & \ln(90) \\ 1 & \ln(100) \\ 1 & \ln(150) \\ 1 & \ln(200) \end{pmatrix} * \begin{pmatrix} \ln(a) \\ b \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \ln(10) & \ln(17) & \ln(20) & \ln(30) & \ln(40) & \ln(50) & \ln(50) & \ln(60) & \ln(70) & \ln(80) & \ln(90) & \ln(100) & \ln(150) & \ln(200) \end{pmatrix} * \begin{pmatrix} 1 & \ln(2{,}3) \\ 1 & \ln(2) \\ 1 & \ln(1{,}85) \\ 1 & \ln(1{,}65) \\ 1 & \ln(1{,}52) \\ 1 & \ln(1{,}4) \\ 1 & \ln(1{,}31) \\ 1 & \ln(1{,}25) \\ 1 & \ln(1{,}19) \\ 1 & \ln(1{,}16) \\ 1 & \ln(1{,}07) \\ 1 & \ln(0{,}91) \\ 1 & \ln(0{,}8) \end{pmatrix}$$

Tras realizar las multiplicaciones se ha obtenido el siguiente sistema con dos incógnitas:

$$\begin{pmatrix} 13 & 51{,}2724 \\ 51{,}2724 & 211{,}4333 \end{pmatrix} * \begin{pmatrix} \ln(a) \\ b \end{pmatrix} = \begin{pmatrix} 3{,}9750 \\ 12{,}4702 \end{pmatrix}$$

$$\left.\begin{matrix} 13\ln(a) + 51{,}2724\, b = 3{,}9750 \\ 51{,}8784\ln(a) + 211{,}4333\, b = 12{,}4702 \end{matrix}\right\}$$

Se ha despejado la primera ecuación y se ha sustituido en la siguiente:

$$\ln(a) = \frac{3{,}9750 - 51{,}2724b}{13}$$

$$\ln(a) = 0{,}3058 - 3{,}9440\, b$$

$$51{,}8784\,(0{,}3058 - 3{,}9440\, b) + 211{,}4333\, b = 12{,}4702$$

$$15{,}6775 - 202{,}2201\, b + 211{,}4333\, b = 12{,}4702$$

$$9{,}2132\, b = -3{,}2074$$

$$\boxed{b = -0{,}3481}$$

Una vez obtenida una incógnita se ha sustituido en una de las ecuaciones anteriores, en este caso en la ecuación despejada, para obtener el valor de la otra incógnita:

$$\ln(a) = 0{,}3058 - 3{,}9440 * (-0{,}3481)$$

$$\ln(a) = 0{,}3058 - (-1{,}373)$$

$$\ln(a) = 1{,}6788$$

$$a = e^{\ln(a)} = e^{1{,}6788}$$

$$\boxed{a = 5{,}5359}$$

Tras obtener las dos incógnitas, la ecuación final sería: $y = 5{,}54 * x^{-0{,}35}$

Una vez obtenida la ecuación exponencial, se ha relacionado con los valores que aparecen en la gráfica del sensor antes comentada, estos valores son:

$$\frac{Rs}{R0} = 5{,}54 * (ppm)^{-0{,}35}$$

En este punto, se pretende despejar $R0$ para obtener su valor calibrado conociendo el valor actual de Co2 medido como ppm y la medida del sensor Rs.

$$R0 = \frac{Rs}{5{,}54 * (ppm)^{-0{,}35}}$$

Por tanto, lo único que se necesita ahora es la medida real de Co2 medido en ppm y la medida del sensor, para la cual se va a realizar la media de n mediciones. Más concretamente, se va a obtener la medida del sensor cada segundo dentro de un rango de 5 minutos y obtener la media. Por lo que Rs parará a ser una media de mediciones.

$$R0 = \frac{Rs\_medio}{5{,}54 * (ppm)^{-0{,}35}}$$

$$Rs\_medio = \frac{\sum_1^n Rs}{n}$$

El sensor de Co2 realmente mide el voltaje o adc y lo manda como señal analógica, por tanto, se debe convertir a señal digital mediante la siguiente formula, siendo Rl la resistencia de carga que según el fabricante es de $20k\Omega = 20000\Omega$

$$Rs = 1024 * \left(\frac{Rl}{adc}\right) - Rl$$

$$Rs = 1024 * \left(\frac{2 * 10^4}{adc}\right) - 2 * 10^4$$

$$Rs = \left(\frac{2048 * 10^4}{adc}\right) - 2 * 10^4$$

Una vez ya se conoce la formula mediante la cual se puede obtener el valor de $R0$ calibrado, este valor actuará como constante para poder obtener el valor de Co2 medido en ppm que está recogiendo el sensor.

El fabricante indica que el tiempo de calibración de este sensor es de alrededor de 24h, por lo que se ha dejado el sistema con el programa de calibración durante ese tiempo para obtener el valor medio de $R0$ que ha sido $R0 = 284.929$. El programa de calibración esta detallado en el Anexo 3.

Para la medición de la calidad del aire en ppm de Co2, solo hay que utilizar la siguiente formula:

$$\frac{Rs}{R0} = a * (ppm)^b$$

$$ppm = \left(\frac{\left(\frac{Rs}{R0}\right)}{a}\right)^{1/b}$$

Conociendo los valores:

$$a = 5{,}5359$$

$$b = -0{,}3481$$

$$R0 = 284929$$

$$Rs = \left(\frac{2048 * 10^4}{adc}\right) - 2 * 10^4$$

$$adc = medida\ del\ senosor$$

Anexo 3 - Programa para calibrar el sensor MQ135

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#define sensorPin A0 //Pin del modulo wifi donde se encuentra el sensor

//Variables del sensor
float a = 5.5359; //  Factor de escala
float b = -0.3481; // Exponente
int Rl = 20000; // Resistencia de carga Rl = 20k(omhio)
float ppm_actual = 418.87; // Valor recogido desde https://www.co2.earth/
int mediciones = 300; // Numero de mediciones cada 5 minutos
float R0; // Resistencia contante que se desea obtener
int adc;  // Lectura de la salida del sensor
int Rs_medio; // Preomedio de las lecturas
float Rs; // Lectura

// Parámetros configuración WiFi
const char* ssid = "SSID";
const char* password = "password";

// Parámetros configuración broker MQTT
const char* brokerUrl = "13.80.129.109";
const char* userMQTT = "monica";
const char* passMQTT = "monica";
const char* topicMQTT = "prueba/co2";
const String clienteId = "Sensor1";

// Instancia al objeto WiFiClient y PubSubClient
WiFiClient clienteESP;
PubSubClient clienteMqtt(clienteESP)

void configuracionWifi() {
  Serial.print("Inicio conexión con la red WiFi ");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);  // Modo estación
  WiFi.begin(ssid, password); // Inicio comunicación
  while (WiFi.status() != WL_CONNECTED) { // Conexión con red WiFi
    delay(500);
  }
}

void obtenerR0() {
  int suma = 0;
  for (int j = 0; j < mediciones; j++){ // Calucular R0 cada segundo durante 5 mininutos
    adc = analogRead(sensorPin);
    Rs = 1024*(Rl/adc)-Rl;
    suma = suma + Rs;
    delay(1000);
  }
  Rs_medio = (suma / mediciones);
  R0 = Rs_medio / (a * pow(ppm_actual, b));
  char payload[7]; // Número de caracteres máximo 6 (XXX.XX)
  snprintf(payload, 7, "%f", R0); // Convertir un float en un array de caracteres
  conectarMQTT();
  clienteMqtt.publish(topicMQTT, payload);  // Enviar mensaje con el valor R0
}

void reconnect() {
  while (!clienteMqtt.connected()) {    // Repetir hasta que se conecte
    Serial.print("Intentando conectarse al broker MQTT...");
    if (clienteMqtt.connect(clienteId.c_str(), userMQTT, passMQTT)) {
      Serial.println("Conectado");
    } else {
      Serial.print("Fallo al conectar al broker, rc=");
      Serial.print(clienteMqtt.state());
      Serial.println(" intentando conectar en 5 segundos");
```

```
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(9600);
  configuracionWifi();  // Configuración WiFi
  clienteMqtt.setServer(brokerUrl, brokerPuerto); // Configuración MQTT
}

void loop() {
  obtenerR0();
}

void conectarMQTT(){
  // Gestión conexión MQTT
  if (!clienteMqtt.connected()) {
    reconnect();
  }
  clienteMqtt.loop();
}
```

Con este programa se pretende conseguir un $R0$ medio después de tener el sensor calibrándose durante 24h. Se podría haber escrito desde el IDE de Arduino los valores de $R0$ obtenidos en un fichero de texto, pero debido a que ya se contaba con la conexión al servidor hecha, simplemente se ha utilizado para escribir los datos en un fichero en el servidor. Además, así se ha podido tener el sensor al aire sin necesidad de estar conectado a una máquina u ordenador y ha podido recoger los datos y mandarlos vía wifi al servidor.

Anexo 4 - Programa para calcular Co2 en ppm y mandarlo al MQTT broker

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#define sensorPin A0 // Pin del modulo wifi donde se encuentra el sensor

//Variables del sensor
float a = 5.5359; //  Factor de escala
float b = -0.3481; // Exponente
int Rl = 12000; // Resistencia de carga Rl = entre 10k y 20k(omhio)
float R0 = 284929.49; // Resistencia contante que se desea obtener
float ppm_calculado; // Calculo del PPM
int mediciones = 60; // Mediciones por minuto
int tiempo_espera = 540000; // Tiempo de espera entre mediciones
int adc;  // Lectura de la salida del sensor
int Rs_medio; // Preomedio de las lecturas
float Rs; // Lectura

// Parámetros configuración WiFi
const char* ssid = "SSID";
const char* password = "password";

// Parámetros configuración broker MQTT
const char* brokerUrl = "13.80.129.109";
int brokerPuerto = 1883;
const char* userMQTT = "monica";
const char* passMQTT = "monica";
const char* topicMQTT = "Co2/Sensor1";
const String clienteId = "Sensor1";

// Instancia al objeto WiFiClient y PubSubClient
WiFiClient clienteESP;
PubSubClient clienteMqtt(clienteESP);

void configuracionWifi() {
  Serial.print("Inicio conexión con la red WiFi ");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);  // Modo estación
  WiFi.begin(ssid, password); // Inicio comunicación
  while (WiFi.status() != WL_CONNECTED) { // Conexión con red WiFi
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Conectado a la red WiFi ");
  Serial.println(ssid);
  Serial.print("Con la IP ");
  Serial.println(WiFi.localIP());
}

void obtenerEnviarCo2() {
  Serial.println("Calculando...");
  int suma = 0;
  for (int j = 0; j < mediciones; j++){ // Recoger valor Rs cada segundo durante 1 minuto
    adc = analogRead(sensorPin);
    Rs = 1024*(Rl/adc)-Rl;
    suma = suma + Rs;
    delay(1000);
  }
  Rs_medio = (suma / mediciones);
  ppm_calculado = pow( ((Rs_medio/R0)/a), (1/b) );
  Serial.print("El valor del Co2 en ppm es: ");
  Serial.println(ppm_calculado);
  char payload[7]; // Número de caracteres máximo 6 (XXX.XX)
  snprintf(payload, 7, "%d", int(ppm_calculado)); // Convertir entero en array de char
  conectarMQTT();
  clienteMqtt.publish(topicMQTT, payload);  // Enviar mensaje con el valor Co2 en ppm
```

```
  Serial.print("Enviado mensaje con Co2 (ppm) ");
  Serial.print(payload);
  Serial.print(" al topic ");
  Serial.println(topicMQTT);
}

void reconnect() {
  while (!clienteMqtt.connected()) {    // Repetir hasta que se conecte
    Serial.print("Intentando conectarse al broker MQTT...");
    if (clienteMqtt.connect(clienteId.c_str(), userMQTT, passMQTT)) {
      Serial.println(" Conectado");
    } else {
      Serial.print("Fallo al conectar al broker, rc=");
      Serial.print(clienteMqtt.state());
      Serial.println(" intentando conectar en 5 segundos");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(9600);
  configuracionWifi();  // Configuración WiFi
  clienteMqtt.setServer(brokerUrl, brokerPuerto); // Configuración MQTT
}

void loop() {
  obtenerEnviarCo2();
  delay(tiempo_espera);
}

void conectarMQTT(){
  // Gestión conexión MQTT
  if (!clienteMqtt.connected()) {
    reconnect();
  }
  clienteMqtt.loop();
}
```

Anexo 5 - Instalación de Docker en una maquina con Ubuntu como sistema operativo

La documentación en la que se ha basado la instalación se encuentra en el enlace [22]. En esta se describen los datos a seguir para instalar Docker en Ubuntu, pero también está la opción de hacerlo en Windows, Mac y otras distribuciones de Linux.

El primer paso es comprobar la versión de Ubuntu instalada en la máquina, en este caso en el servidor de Azure.

```
azureuser@azureserver:~$ cat /etc/*release*
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.5 LTS"
NAME="Ubuntu"
VERSION="18.04.5 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.5 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
```

Ilustración 23: Comprobar configuración Ubuntu

El siguiente paso es desinstalar las versiones anteriores de Docker, si están instaladas en la máquina, para asegurarse de instalar la versión más actualizada. En el caso actual no existe Docker por lo que no encuentra los paquetes a desinstalar.

```
azureuser@azureserver:~$ sudo apt-get remove docker docker-engine docker.io containerd runc
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package docker
E: Unable to locate package docker-engine
E: Unable to locate package docker.io
E: Couldn't find any package by glob 'docker.io'
E: Couldn't find any package by regex 'docker.io'
E: Unable to locate package containerd
E: Unable to locate package runc
```

Ilustración 24: Comando para desinstalar Docker

El siguiente paso es instalar Docker, dentro de la documentación existen tres métodos de instalación:

1. Configurar los repositorios e instalar Docker desde ellos.
2. Descargar el paquete DEB e instalarlo manualmente.
3. Utilizar un script automatizado.

Esta última opción se suele utilizar en entornos de prueba y desarrollo y además es la más sencilla, por tanto, ha sido la forma escogida para instalar Docker.

Los pasos para seguir son descargar el script de https://get.docker.com y ejecutarlo.

```
azureuser@azureserver:~$ curl -fsSL https://get.docker.com -o get-docker.sh
azureuser@azureserver:~$ sudo sh get-docker.sh
# Executing docker install script, commit: 7cae5f8b0decc17d6571f9f52eb840fbc13b2737
```

Ilustración 25: Comando para descargar el script y ejecutarlo

Tras estos sencillos pasos Docker está instalado en la máquina y para comprobarlo simplemente es necesario comprobar la versión de Docker con el siguiente comando.

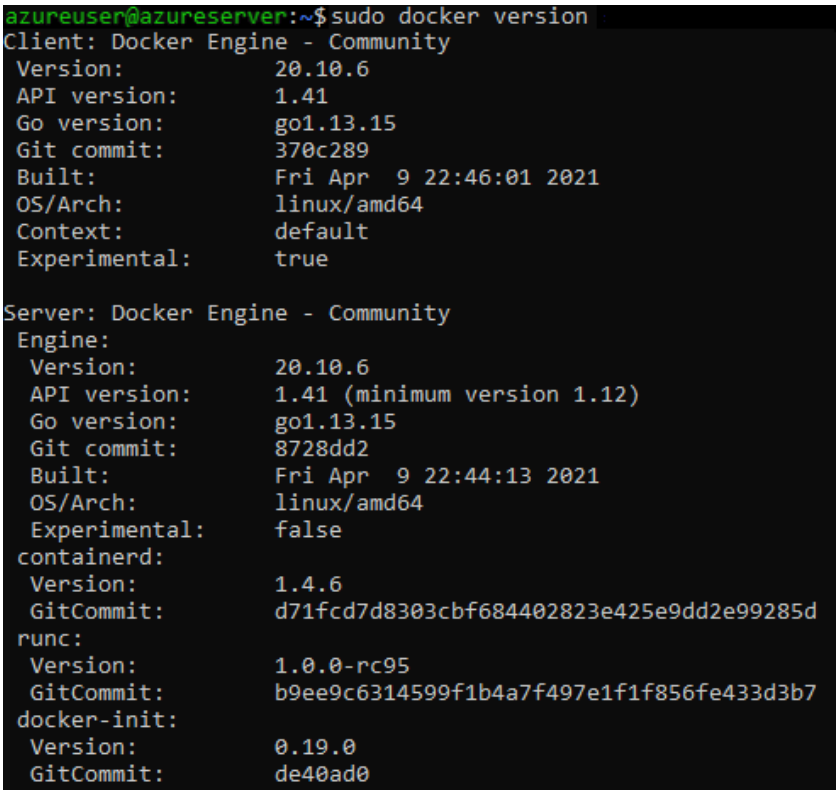

Ilustración 26: Comprobar configuración Docker

Anexo 6 - Instalación de la imagen de Mosquitto y primeras configuraciones

Para esta instalación se ha tenido en cuenta la posibilidad de que el contenedor que va a albergar Mosquitto pueda tener problemas y para no perder los datos, el log y la configuración se han creado volúmenes para guardar esta información en local. Para ello se han creado las tres carpetas antes mencionadas, que son las mismas carpetas donde guarda Mosquitto su información.

```
azureuser@azureserver:~$ mkdir mosquitto
azureuser@azureserver:~$ cd mosquitto/
azureuser@azureserver:~/mosquitto$ mkdir config
azureuser@azureserver:~/mosquitto$ mkdir data
azureuser@azureserver:~/mosquitto$ mkdir log
azureuser@azureserver:~/mosquitto$ ls
config  data  log
```

Ilustración 27: Directorios donde se guarda la información de Mosquitto en local

También se ha creado un archivo de configuración para Mosquitto en el que aparecen:

- El puerto en el que debe escuchar la conexión de red. Está vinculada a la IP 0.0.0.0 que hace referencia a todas las direcciones IP.
- El archivo PID, donde se guardará el identificador del proceso.
- La configuración de almacenamiento persistente que guardar los datos en el directorio descrito en persistance_location.
- El directorio donde internamente se va a guardar la información.
- El archivo de registros .log donde se guardarán los registros de la aplicación.
- El archivo de contraseña .passwd donde irá a buscar la contraseña para acceder a la aplicación.
- La configuración de permitir accesos anónimos.

```
azureuser@azureserver:~/mosquitto/config$ cat mosquitto.conf
listener 1883 0.0.0.0
pid_file /var/run/mosquitto.pid
persistence true
persistence_location /mqtt/data
log_dest file /mqtt/mosquitto.log
password_file /mqtt/config/mosquitto.passwd
allow_anonymous true
```

Ilustración 28: Archivo mosquito.conf

Al tratarse de un proyecto que va a utilizar más de una imagen, se ha creado un Docker compose desde el que se ejecutarán todas las imágenes. Se ha creado un directorio denominado "compose" donde crear el archivo con la configuración de Mosquitto.

```
azureuser@azureserver:~$ mkdir compose
azureuser@azureserver:~$ cd compose/
azureuser@azureserver:~/compose$ cat docker-compose.yml
version: '3'
services:
##################################################MOSQUITTO
    mqtt:
        container_name: mosquitto
        image: eclipse-mosquitto:latest
        ports:
            - "1833:1833"
            - "9001:9001"
        volumes:
            - /home/azureuser/mosquitto/config:/mqtt/config
            - /home/azureuser/mosquitto/log:/mqtt/log
            - /home/azureuser/mosquitto/data:/mqtt/data
```

Ilustración 29: Archivo docker-compose.yml

En este archivo se describe la versión del “Docker compose” y el servicio Mosquitto. Este servicio consta de un nombre de contenedor, una imagen que se desea cargar, unos puertos y unos volúmenes. Los puertos descritos se mapearán desde la red del contenedor hasta la red del Docker Host, es decir, lo que se reciba por el puerto 1883 de la IP del Docker Host irá directamente al puerto 1883 del contenedor y viceversa, lo mismo con el puerto 9001. En el caso de los volúmenes lo que se describe es que se debe guardar la información que el contenedor tiene en su directorio /mqtt/ de forma local en el directorio /home/azureuser/mosquitto.

Después de completar el archivo docker-compose.yml con la configuración de Mosquitto, se ha instalado Docker compose para poder ejecutar el archivo.

```
azureuser@azureserver:~/compose$ sudo apt install docker-compose
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Ilustración 30: Comando para instalar “Docker compose”

Se ha ejecutado el Docker compose. Como ha sido la primera vez que se ha ejecutado, se ha creado una red por defecto, porque no se había descrito ninguna, se ha descargado la imagen de Mosquitto del Docker Hub, porque no se disponía ya de ella, y se ha creado el contenedor con el nombre descrito en el archivo.

```
azureuser@azureserver:~/compose$ sudo docker-compose up -d
Creating network "compose_default" with the default driver
Pulling mqtt (eclipse-mosquitto:latest)...
latest: Pulling from library/eclipse-mosquitto
339de151aab4: Pull complete
8cc7b833e2db: Pull complete
f2b454175e7e: Pull complete
Digest: sha256:683189aaaf01240fe4ba3c0b3262704c4a3910c7293162e22e8f43a319fc9ed8
Status: Downloaded newer image for eclipse-mosquitto:latest
Creating mosquitto ...
Creating mosquitto ... done
```

Ilustración 31: Ejecutar el Docker compose

A continuación, se ha comprobado que efectivamente el contenedor de Mosquitto está corriendo en el servidor dentro del Docker y se ha entrado a su terminal para crear el archivo de contraseñas.

```
azureuser@azureserver:~/compose$ sudo docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS
                                                                                NAMES
f949a455cb70   eclipse-mosquitto:latest   "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes   0.0.0.0:1833->1833/tcp
, :::1833->1833/tcp, 0.0.0.0:9001->9001/tcp, :::9001->9001/tcp, 1883/tcp   mosquitto
azureuser@azureserver:~/compose$ sudo docker exec -it mosquitto sh
/ # ls
bin                   etc                   media                 mosquitto-no-auth.conf  proc
sbin                  tmp
dev                   home                  mnt                   mqtt                    root
srv                   usr
docker-entrypoint.sh  lib                   mosquitto             opt                     run
sys                   var
/ # mosquitto_passwd -c /mqtt/config/mosquitto.passwd monica
Password:
Reenter password:
/ # exit
```

Ilustración 32: Acceder al terminal del contenedor y crear el archivo mosquitto.passwd

No se puede concluir que Mosquitto está bien configurado hasta que no se prueba. Para ello se han abierto dos terminales dentro del contenedor, una de ellas como usuario que publica datos en un tema y el otro como usuario suscrito a este mismo tema. Se ha probado a enviar mensajes públicos y privados, para lo que es necesario disponer de un usuario y una contraseña.

```
azureuser@azureserver:~/compose$ sudo docker exec -it mosquitto sh
/ # mosquitto_pub -t "test/topic" -m "Hello World!"
/ # mosquitto_pub -t "test/topic" -m "Hola Mónica"
/ # mosquitto_pub -t "test/topic" -m "Mosquitto funciona"
/ # exit
```

Ilustración 33: Usuario que publica en el tema "test/topic"

```
azureuser@azureserver:~/compose$ sudo docker exec -it mosquitto sh
/ # mosquitto_sub -v -t "test/topic"
test/topic Hello World!
test/topic Hola Mónica
test/topic Mosquitto funciona
```

Ilustración 34: Usuario suscrito en el tema "test/topic"

```
azureuser@azureserver:~/compose$ sudo docker exec -it mosquitto sh
/ # mosquitto_pub -d -u "username" -P "password" -t "test/password" -m "Mensaje password"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q0, r0, m1, 'test/password', ... (16 bytes))
Client (null) sending DISCONNECT
/ # exit
```

Ilustración 35: Usuario que publica en el tema "test/password" con usuario y contraseña

```
azureuser@azureserver:~/compose$ sudo docker exec -it mosquitto sh
/ # mosquitto_sub -d -u "username" -P "password" -t "test/password"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: test/password, QoS: 0, Options: 0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
Client (null) sending PINGREQ
Client (null) received PINGRESP
Client (null) received PUBLISH (d0, q0, r0, m0, 'test/password', ... (16 bytes))
Mensaje password
```

Ilustración 36: Usuario suscrito en el tema "test/ password" con usuario y contraseña

Como se puede apreciar en las imágenes, Mosquitto está configurado correctamente porque el usuario suscriptor ha recibido los mensajes publicados por el otro usuario en ambos casos.

Anexo 7 - Instalación de la imagen de Node-red y primeras configuraciones

Para esta instalación se ha tenido en cuenta la posibilidad de que el contenedor que va a albergar Node-red pueda tener problemas y para no perder los datos, el log y la configuración se han creado volúmenes para guardar esta información en local. Para ello se ha creado un directorio que es el mismo donde Node-red guarda su información.

```
azureuser@azureserver:~$ mkdir node-red
azureuser@azureserver:~$ cd node-red/
```

Ilustración 37: Directorios y archivos donde se guarda la información de Node-red en local

Al tratarse de un proyecto que va a utilizar más de una imagen, se ha creado un Docker compose, ampliando el mostrado en el Anexo 6, desde el que se ejecutarán ambas imágenes. Se ha ampliado la parte de configuración de Node-red quedando el archivo de la siguiente forma:

```
azureuser@azureserver:~/compose$ cat docker-compose.yml
version: '3'
services:
##################################################MOSQUITTO
    mqtt:
        container_name: mosquitto
        image: eclipse-mosquitto:latest
        ports:
            - "1833:1833"
            - "9001:9001"
        volumes:
            - /home/azureuser/mosquitto/config:/mqtt/config
            - /home/azureuser/mosquitto/log:/mqtt/log
            - /home/azureuser/mosquitto/data:/mqtt/data
##################################################NODE-RED
    nodered:
        container_name: nodered
        image: nodered/node-red
        ports:
            - "1880:1880"
        restart: always
        volumes:
            - /home/azureuser/node-red:/data
```

Ilustración 38: Archivo docker-compose.yml

En este archivo se describe la versión del "Docker compose", el servicio Mosquitto, descrito en el Anexo 6 y el servicio Node-red. Este último servicio consta de un nombre de contenedor, una imagen que se desea cargar, un puerto y un volumen. El puerto descrito se mapeará desde la red del contenedor hasta la red del Docker Host, es decir, lo que se reciba por el puerto 1880 de la IP del Docker Host irá directamente al puerto 1880 del contenedor y viceversa. En el caso del volumen lo que se describe es que se debe guardar la información que el contenedor tiene en su directorio /data de forma local en el directorio /home/azureuser/node-red.

Después de completar el archivo docker-compose.yml con la configuración de Node-red, se ha ejecutado el mismo para obtener la ejecución de las dos imágenes. Como ha sido la primera vez que se ha ejecutado con la imagen de Node-red, se ha descargado esta imagen, porque no se disponía de ella y se ha creado el contenedor con el nombre descrito en el archivo. Además, comenta que el contenedor de Mosquitto está actualizado ya que se encontraba ejecutándose, si no hubiera sido así, se hubiera creado.

```
azureuser@azureserver:~/compose$ sudo docker-compose up -d
Pulling nodered (nodered/node-red:latest)...
latest: Pulling from nodered/node-red
ddad3d7c1e96: Pull complete
de915e575d22: Pull complete
7150aa69525b: Pull complete
d7aa47be044e: Pull complete
6534de536318: Pull complete
674367ed9b4e: Pull complete
c6478b35f017: Pull complete
353e61167e7c: Pull complete
2f35e060331e: Pull complete
f3bd251d9b78: Pull complete
a3ff093ffcc8: Pull complete
779c22000020: Pull complete
Digest: sha256:f7a50c7c144d4472ead0e9eda8a7447aa3df6002e7851dfc7f5ba774b9598cd2
Status: Downloaded newer image for nodered/node-red:latest
Creating nodered ...
mosquitto is up-to-date
Creating nodered ... done
```

Ilustración 39: Ejecutar el Docker compose

A continuación, se dispone de un contenedor con Mosquitto y otro con Node-red corriendo dentro del Docker.

No se puede concluir que Node-red está bien configurado hasta que no se pruebe. Para ello se ha configurado en la máquina virtual, en las reglas de seguridad, que el puerto 1880 permita conexiones desde cualquier IP, también se ha añadido el puerto 1883 que se utilizará más adelante.

120	Port_1880	1880	Cualquiera	Cualquiera	Cualquiera	Allow
130	Port_1883_mqtt	1883	Cualquiera	Cualquiera	Cualquiera	Allow

Por tanto, para acceder a la interfaz de usuario de Node-red, se ha abierto un navegador web y colocado como URL la IP de la máquina virtual y el puerto 1880 consiguiendo la siguiente pantalla:

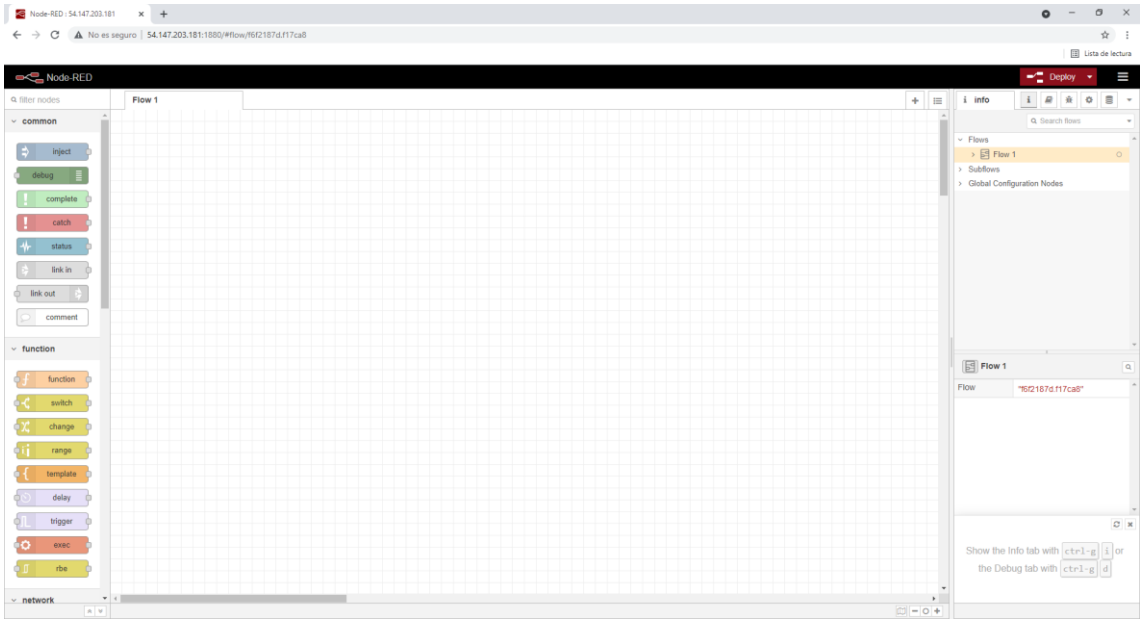

Ilustración 40: Interfaz de usuario Node-red.

En este punto se puede acceder a Mosquitto y Node-red desde el exterior del servidor. Por tanto, para conectarlos entre sí, Node-red va a acceder a través de la IP del servidor y el puerto de Mosquitto, 1883 previamente configurado, al broker MQTT.

Para probar la comunicación entre ambos contenedores se ha creado en Node-red un suscriptor del tema “test/topic” y un publicador del tema “test/topic2”. Para ello se han arrastrado los nodos necesarios, se ha añadido su configuración pertinente y se ha pulsado el botón Deploy para desplegar la construcción.

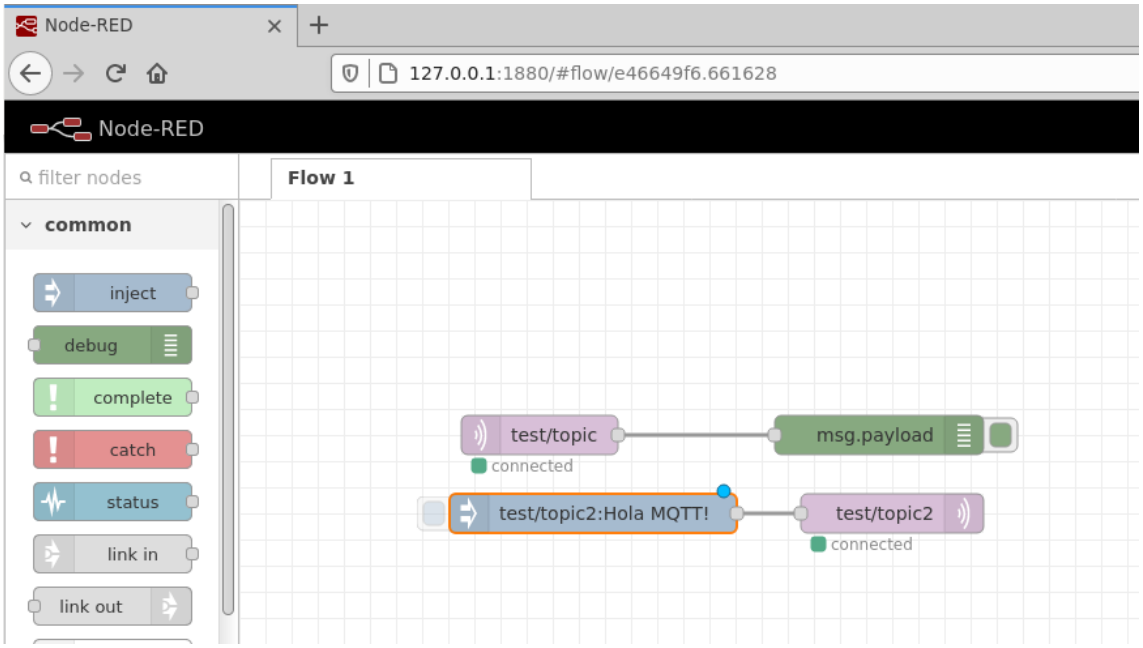

Ilustración 41: Creando subscriptor y publicador de Mosquitto en Node-red

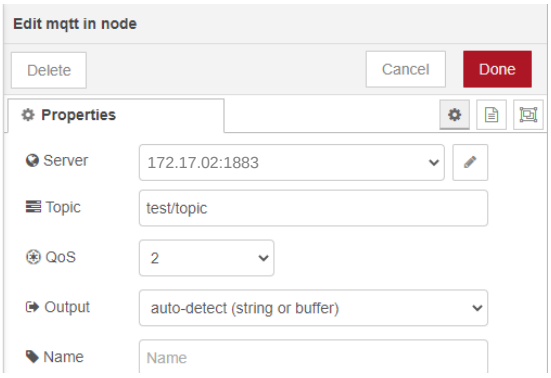

Ilustración 42: Configuración del subscriptor de Mosquitto en Node-red

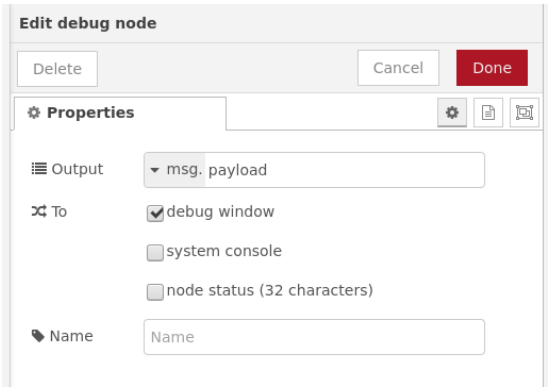

Ilustración 43: Configuración del nodo msg.payload

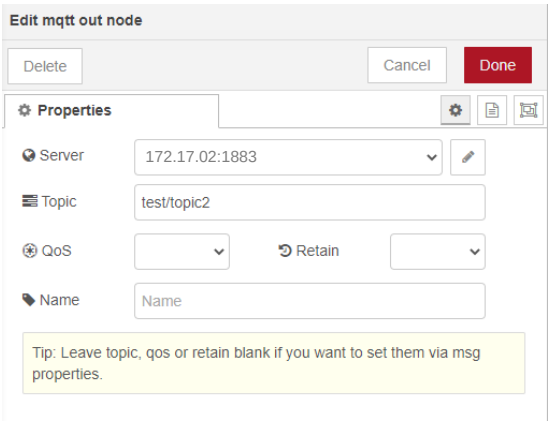

Ilustración 44: Configuración del publicador de Mosquitto en Node-red

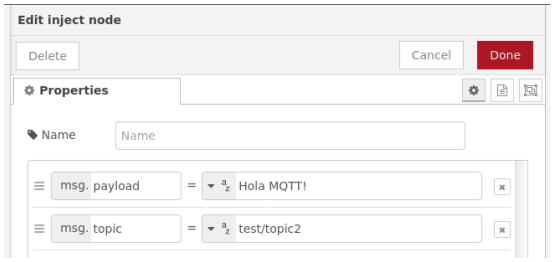

Ilustración 45: Configuración del nodo "test/topic2:Hola MQTT!"

Ilustración 46: Botón Deploy que despliega el flujo de Node-red

Después de esto, para probar la configuración de la construcción en Node-red, se han abierto dos terminales dentro del contenedor de Mosquitto, una de ellas como publicador del tema "test/topic" y la otra como suscriptor del tema "test/topic2".

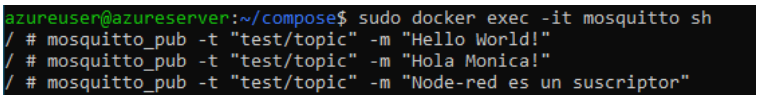

Ilustración 47: Usuario que publica en el tema "test/topic" en Mosquitto

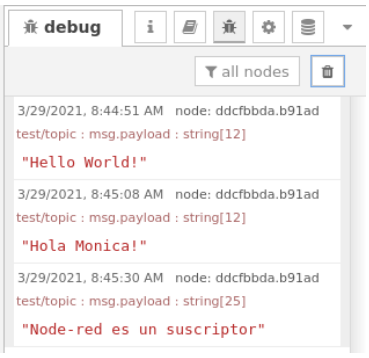

Ilustración 48: Resultado obtenido en el suscriptor de Node-red

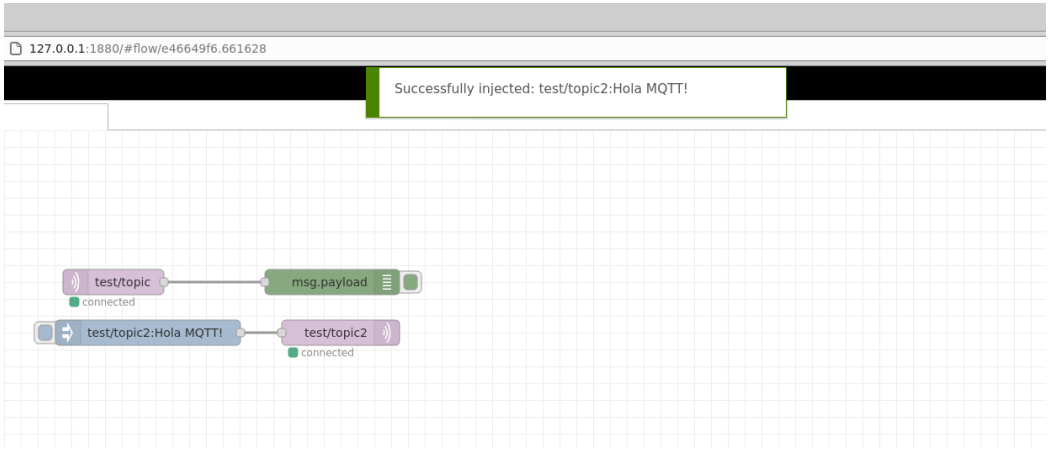

Ilustración 49: Resultado obtenido al enviar mensaje a través de Node-red en el tema "test/topic2"

```
azureuser@azureserver:~$ sudo docker exec -it mosquitto sh
/ # mosquitto_sub -v -t "test/topic2"
test/topic2 Hola MQTT!
test/topic2 Hola MQTT!
test/topic2 Hola MQTT!
```

Ilustración 50: Usuario suscrito en el tema "test/topic2" en Mosquitto

Como se puede apreciar en las imágenes, los mensajes publicados en el tema "test/topic" por la terminal de Mosquitto son recibidos por el suscriptor de Node-red y los mensajes publicados en el tema "test/topic2" por el publicador de Node-red son recibidos por la terminal de Mosquitto. Por tanto, se puede concluir que Node-red está configurado correctamente y funciona como publicador y suscriptor del broker MQTT Mosquito.

Anexo 8 - Requisitos obtenidos en la etapa de análisis

Código	Nombre	Prioridad	Descripción
01	Listado de sensores	Media	Listado de sensores con filtro
02	Seleccionar sensor	Media	Seleccionar un sensor para ver su información
03	Listado de datos por sensor	Media	Listado de los datos de un sensor
04	Editar sensor	Media	Editar la información de un sensor
05	Añadir sensor	Baja	Añadir un sensor con su información
06	Eliminar sensor	Baja	Eliminar sensor de base se datos
04	Lista de alertas	Alta	Listado de alertas con filtro
07	Gráficos de alertas	Alta	Gráficos con los datos de todas las alertas
08	Seleccionar alerta	Alta	Seleccionar una alerta para ver su información
09	Listado de datos por alerta	Alta	Listado de los datos de una alerta
10	Editar alerta	Media	Editar la información de una alerta
11	Añadir alerta	Alta	Añadir una alerta con su información
12	Eliminar alerta	Media	Eliminar alerta de la base de datos
13	Listado de datos	Alta	Listado de datos con filtro
14	Gráficos de datos	Alta	Gráficos con todos los datos y con filtro
15	Perfil	Alta	Perfil del usuario con su información
16	Editar perfil	Alta	Editar la información de un usuario
17	Lista de usuarios	Media	Lista de usuarios
18	Seleccionar usuario	Media	Seleccionar un usuario para ver su información
19	Añadir usuarios	Media	Añadir un usuario
20	Editar usuario	Baja	Editar la información de un usuario
21	Eliminar usuario	Baja	Eliminar un usuario de la base de datos

Tabla 15: Requisitos funcionales

Código	Nombre	Prioridad	Descripción
22	Base de datos	Alta	La aplicación debe funcionar sobre una base de datos Microsoft SQL server 2017.
23	Documentación	Alta	El sistema debe estar debidamente documentado.
24	Rapidez	Media	El sistema debe actualizar los datos de los sensores de forma rápida.

Tabla 16: Requisitos no funcionales

Anexo 9 - Casos de uso obtenidos en la etapa de análisis

Caso de uso 1: Listado de sensores

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de sensores
Postcondiciones	El usuario ha visto el listado de sensores.
Flujo principal	1. El usuario está en la pantalla de sensores y puede: 1.1 Ver el listado de sensores. 1.2 Utilizar los filtros de búsqueda. 2. El sistema consulta la base de datos con los filtros en caso de estar activados. 3. La aplicación muestra, a través de una tabla, el listado de sensores.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error.
Frecuencia	Muy frecuente.

Caso de uso 2: Seleccionar sensor

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de sensores.
Postcondiciones	El usuario ha visto el detalle de un sensor.
Flujo principal	1. El usuario está en la pantalla del listado de sensores y pulsa el botón de visualizar los datos de un sensor. 2. El sistema consulta la base de datos. 3. La aplicación muestra los datos del sensor. 4. El usuario ahora puede decidir entre: 4.1 Ver un listado con los datos recabados por el sensor. 4.1.1 El sistema vuelve a consultar la base de datos. 4.1.2 La aplicación muestra una tabla con los datos. 4.2 Ver una gráfica con los datos recabados por el sensor en un día concreto sobre un filtro. 4.2.1 El sistema vuelve a consultar la base de datos. 4.2.2 La aplicación muestra una gráfica con los datos del día seleccionado. 4.3 Volver a la pantalla anterior. 4.3.1 El sistema consulta la base de datos. 4.3.2 La aplicación muestra el listado de sensores.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error.
Frecuencia	Muy frecuente.

Caso de uso 3: Editar sensor

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del detalle del sensor.
Postcondiciones	El usuario ha editado los datos del sensor.
Flujo principal	1. El usuario está en la pantalla de detalles de un sensor y pulsa el botón de editar sensor. 2. El sistema consulta la base de datos. 3. La aplicación muestra los datos del sensor que son editables. 4. El usuario ahora puede decidir entre: 4.1 Editar los datos y guardar el sensor. 4.1.1 El sistema modifica la base de datos. 4.2 Volver a la pantalla anterior. 4.2.1 El sistema consulta la base de datos. 5. La aplicación muestra los datos del sensor, en caso de haberlo editado estos datos se verán modificados.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error. 2. El usuario no completa los datos obligatorios. 2.1 La aplicación informa al usuario que datos son obligatorios para poder editar el sensor. 3. El usuario modifica el nombre del sensor a uno existente. 3.1 El sistema comprueba si el nombre existe en la base de datos. 3.2 La aplicación informa al usuario que ese nombre no es válido porque ya existe.
Frecuencia	Media.

Caso de uso 4: Añadir sensor

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del detalle del sensor.
Postcondiciones	El usuario ha creado un nuevo sensor.
Flujo principal	1. El usuario está en la pantalla del listado de sensores y pulsa el botón de añadir un nuevo sensor. 2. La aplicación muestra un formulario con los datos del sensor a rellenar. 3. El usuario ahora puede decidir entre: 3.1 Rellenar los campos y crear el sensor. 3.1.1 El sistema añade el sensor a la base de datos. 3.1.2 La aplicación muestra la pantalla del detalle del nuevo sensor. 3.2 Volver a la pantalla anterior. 3.2.1 El sistema consulta la base de datos. 3.2.2 La aplicación muestra el listado de sensores.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error. 2. El usuario no completa los datos obligatorios. 2.1 La aplicación informa al usuario que datos son obligatorios para poder crear el sensor. 3. El usuario añade un nombre ya existente. 3.1 El sistema comprueba si el nombre existe en la base de datos. 3.2 La aplicación informa al usuario que ese nombre no es válido porque ya existe.
Frecuencia	Poco frecuente.

Caso de uso 5: Eliminar sensor

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del detalle del sensor.
Postcondiciones	El usuario ha eliminado un sensor.
Flujo principal	1. El usuario está en la pantalla del detalle del sensor y pulsa el botón de eliminar el sensor. 2. La aplicación muestra una ventana emergente para que el usuario confirme que desea eliminar el sensor. 3. El usuario ahora puede decidir entre: 3.1 Eliminar el sensor. 3.1.1 El sistema elimina el sensor y los datos relacionados a el de la base de datos. 3.1.2 La aplicación muestra la pantalla del listado de sensores sin el sensor eliminado. 3.2 Cancelar. 3.2.1 La aplicación cierra la ventana emergente dejando ver el detalle del sensor.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error.
Frecuencia	Poco frecuente.

Caso de uso 6: Listado de alertas

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de alertas.
Postcondiciones	El usuario ha visto el listado de alertas.
Flujo principal	1. El usuario está en la pantalla del listado de alarmas y puede: 1.1 Ver el listado de alarmas. 1.2 Utilizar los filtros de búsqueda. 2. El sistema consulta la base de datos con los filtros en caso de estar activados. 3. La aplicación muestra, a través de una tabla, el listado de alarmas.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error.
Frecuencia	Muy frecuente.

Caso de uso 7: Gráficos de alertas

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de alertas.
Postcondiciones	El usuario ha visto el gráfico de alertas.
Flujo principal	1. El usuario está en la pantalla del listado de alarmas y puede: 1.1 Ver el grafico de alarmas. 1.2 Utilizar los filtros de búsqueda. 2. El sistema consulta la base de datos con los filtros en caso de estar activados. 3. La aplicación muestra, a través de una gráfica, el listado de alarmas.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error.
Frecuencia	Muy frecuente.

Caso de uso 8: Seleccionar alerta

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de alertas.
Postcondiciones	El usuario ha visto el detalle de una alerta.
Flujo principal	1. El usuario está en la pantalla de alertas y pulsa el botón de visualizar los datos de una alerta. 2. El sistema consulta la base de datos. 3. La aplicación muestra los datos de la alerta. 4. El usuario ahora puede decidir entre: 4.1 Volver a la pantalla anterior. 4.1.1 El sistema vuelve a consultar la base de datos. 4.1.2 La aplicación muestra todas las alarmas. 4.2 Filtrar el listado de mediciones que han encendido la alerta. 4.2.1 El sistema vuelve a consultar la base de datos. 4.2.2 La aplicación muestra la lista de mediciones filtrada.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error.
Frecuencia	Muy frecuente.

Caso de uso 9: Editar alerta

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del detalle de la alerta.
Postcondiciones	El usuario ha editado los datos de la alerta.
Flujo principal	1. El usuario está en la pantalla de detalles de una alerta y pulsa el botón de editar alerta. 2. El sistema consulta la base de datos. 3. La aplicación muestra los datos de la alerta que son editables. 4. El usuario ahora puede decidir entre: 4.1 Editar los datos y guardar la alarma. 4.1.1 El sistema modifica la base de datos. 4.2 Volver a la pantalla anterior. 4.2.1 El sistema consulta la base de datos. 5. La aplicación muestra los datos de la alarma, en caso de haberla editado estos datos se verán modificados.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error. 2. El usuario no completa los datos obligatorios. 2.1 La aplicación informa al usuario que datos son obligatorios para poder editar la alerta. 3. El usuario modifica el nombre de la alerta a uno existente. 3.1 El sistema comprueba si el nombre existe en la base de datos. 3.2 La aplicación informa al usuario que ese nombre no es válido porque ya existe.
Frecuencia	Media.

Caso de uso 10: Añadir alerta

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de alertas.
Postcondiciones	El usuario ha creado una nueva alerta.
Flujo principal	1. El usuario está en la pantalla de alertas y pulsa el botón de añadir una nueva alerta. 2. La aplicación muestra un formulario con los datos de la alerta a rellenar. 3. El usuario ahora puede decidir entre: 3.1 Rellenar los campos y crear la alerta. 3.1.1 El sistema añade la alerta a la base de datos. 3.1.2 La aplicación muestra la pantalla del detalle de la nueva alerta. 3.2 Volver a la pantalla anterior. 3.2.1 El sistema consulta la base de datos. 3.2.2 La aplicación muestra el listado de alertas.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error. 2. El usuario no completa los datos obligatorios. 2.1 La aplicación informa al usuario que datos son obligatorios para poder crear la alerta. 3. El usuario añade un nombre ya existente. 3.1 El sistema comprueba si el nombre existe en la base de datos. 3.2 La aplicación informa al usuario que ese nombre no es válido porque ya existe.
Frecuencia	Poco frecuente

Caso de uso 11: Eliminar alerta

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del detalle de la alerta.
Postcondiciones	El usuario ha eliminado una alerta.
Flujo principal	1. El usuario está en la pantalla del detalle de la alerta y pulsa el botón de eliminar la alerta. 2. La aplicación muestra una ventana emergente para que el usuario confirme que desea eliminar la alerta. 3. El usuario ahora puede decidir entre: 3.1 Eliminar la alerta. 3.1.1 El sistema elimina la alerta y los datos relacionados a ella de la base de datos. 3.1.2 La aplicación muestra la pantalla del listado de alertas sin la alerta eliminada. 3.2 Cancelar. 3.2.1 La aplicación cierra la ventana emergente dejando ver el detalle de la alerta.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error.
Frecuencia	Poco frecuente.

Caso de uso 12: Listado de datos

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de datos.
Postcondiciones	El usuario ha visto el listado de datos.
Flujo principal	1. El usuario está en la pantalla del listado de datos y puede: 1.1 Ver el listado de datos. 1.2 Utilizar los filtros de búsqueda. 2. El sistema consulta la base de datos con los filtros en caso de estar activados. 3. La aplicación muestra, a través de una tabla, el listado de datos.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error.
Frecuencia	Muy frecuente.

Caso de uso 13: Gráficos de datos

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de datos.
Postcondiciones	El usuario ha visto el gráfico de datos.
Flujo principal	1. El usuario está en la pantalla del listado de datos y puede: 1.1 Ver el grafico de datos. 1.2 Utilizar los filtros de búsqueda. 2. El sistema consulta la base de datos con los filtros en caso de estar activados. 3. La aplicación muestra, a través de una gráfica, el listado de datos.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error.
Frecuencia	Muy frecuente.

Caso de uso 14: Ver perfil

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del perfil.
Postcondiciones	El usuario ha visto el perfil.
Flujo principal	1. El usuario está en la pantalla del perfil. 2. El sistema consulta la base de datos. 3. La aplicación muestra los datos del perfil del usuario.
Flujos alternativos	2. Fallo en el servidor o en la comunicación con la base de datos. 2.1 La aplicación informa al usuario del error.
Frecuencia	Frecuente.

Caso de uso 15: Editar perfil

Actores	Usuario.
Precondiciones	Haber iniciado sesión y estar en la pantalla del perfil.
Postcondiciones	El usuario ha editado el perfil.
Flujo principal	1. El usuario está en la pantalla del perfil y pulsa el botón de editar perfil. 2. El sistema consulta la base de datos. 3. La aplicación muestra los datos del perfil que son editables. 4. El usuario ahora puede decidir entre: 4.1 Editar los datos y guardar el perfil. 4.1.1 El sistema modifica la base de datos. 4.2 Volver a la pantalla anterior. 4.2.1 El sistema consulta la base de datos. 5. La aplicación muestra los datos del perfil, en caso de haberlo editado estos datos se verán modificados.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al usuario del error.
Frecuencia	Poco frecuente.

Caso de uso 16: Listado de usuarios

Actores	Administrador.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de usuarios.
Postcondiciones	El administrador ha visto el listado de usuarios.
Flujo principal	1. El administrador está en la pantalla del listado de usuarios y puede: 1.1 Ver el listado de datos. 1.2 Utilizar los filtros de búsqueda. 2. El sistema consulta la base de datos con los filtros en caso de estar activados. 3. La aplicación muestra, a través de una tabla, el listado de usuarios.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al administrador del error.
Frecuencia	Muy frecuente.

Caso de uso 17: Seleccionar usuario

Actores	Administrador.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de usuarios.
Postcondiciones	El administrador ha visto el detalle de un usuario.
Flujo principal	1. El administrador está en la pantalla del listado de usuarios y pulsa el botón de visualizar los datos de un usuario. 2. El sistema consulta la base de datos. 3. La aplicación muestra los datos del usuario. 4. El administrador ahora puede volver a la pantalla anterior. 4.1 El sistema consulta la base de datos. 4.2 La aplicación muestra el listado de sensores.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al administrador del error.
Frecuencia	Frecuente.

Caso de uso 18: Añadir usuario

Actores	Administrador.
Precondiciones	Haber iniciado sesión y estar en la pantalla del listado de usuarios.
Postcondiciones	El administrador ha creado un nuevo usuario.
Flujo principal	1. El administrador está en la pantalla de usuarios y pulsa el botón de añadir un nuevo usuario. 2. La aplicación muestra un formulario con los datos del usuario a rellenar. 3. El administrador ahora puede decidir entre: 3.1 Rellenar los campos y crear el usuario. 3.1.1 El sistema añade el usuario a la base de datos. 3.1.2 La aplicación muestra la pantalla del detalle del nuevo usuario. 3.2 Volver a la pantalla anterior. 3.2.1 El sistema consulta la base de datos. 3.2.2 La aplicación muestra el listado de usuarios.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al administrador del error. 2. El administrador no completa los datos obligatorios. 2.1 La aplicación informa al administrador que datos son obligatorios para poder crear la alerta. 3. El administrador añade un nombre ya existente. 3.1 El sistema comprueba si el nombre existe en la base de datos. 3.2 La aplicación informa al administrador que ese nombre no es válido porque ya existe.
Frecuencia	Poco frecuente.

Caso de uso 19: Editar usuario

Actores	Administrador.
Precondiciones	Haber iniciado sesión y estar en la pantalla del detalle del usuario.
Postcondiciones	El administrador ha editado el usuario.
Flujo principal	1. El usuario está en la pantalla del usuario y pulsa el botón de editar el usuario. 2. El sistema consulta la base de datos. 3. La aplicación muestra los datos del usuario que son editables. 4. El administrador ahora puede decidir entre: 4.1 Editar los datos y guardar el usuario. 4.1.1 El sistema modifica la base de datos. 4.2 Volver a la pantalla anterior. 4.2.1 El sistema consulta la base de datos. 5. La aplicación muestra los datos del usuario, en caso de haberlo editado estos datos se verán modificados.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al administrador del error.
Frecuencia	Poco frecuente.

Caso de uso 20: Eliminar usuario

Actores	Administrador.
Precondiciones	Haber iniciado sesión y estar en la pantalla del detalle del usuario.
Postcondiciones	El usuario ha eliminado un usuario.
Flujo principal	1. El administrador está en la pantalla del detalle del usuario y pulsa el botón de eliminar el usuario. 2. La aplicación muestra una ventana emergente para que el usuario confirme que desea eliminar el usuario. 3. El administrador ahora puede decidir entre: 3.1 Eliminar el usuario. 3.1.1 El sistema elimina el usuario y los datos relacionados a el de la base de datos. 3.1.2 La aplicación muestra la pantalla del listado de usuarios sin el usuario eliminado. 3.2 Cancelar. 3.2.1 La aplicación cierra la ventana emergente dejando ver el detalle del usuario.
Flujos alternativos	1. Fallo en el servidor o en la comunicación con la base de datos. 1.1 La aplicación informa al administrador del error.
Frecuencia	Poco frecuente.

Anexo 10 - Pantallas obtenidas en la etapa de diseño

Ilustración 51: diseño del inicio de sesión o Login

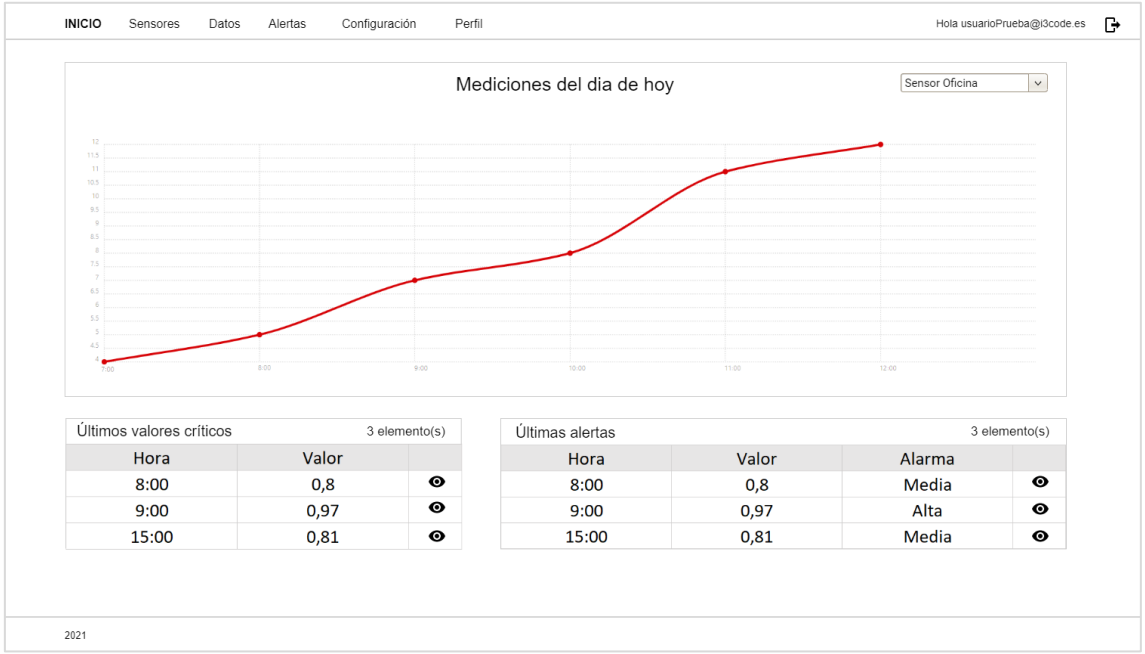

Ilustración 52: Diseño de pantalla de inicio

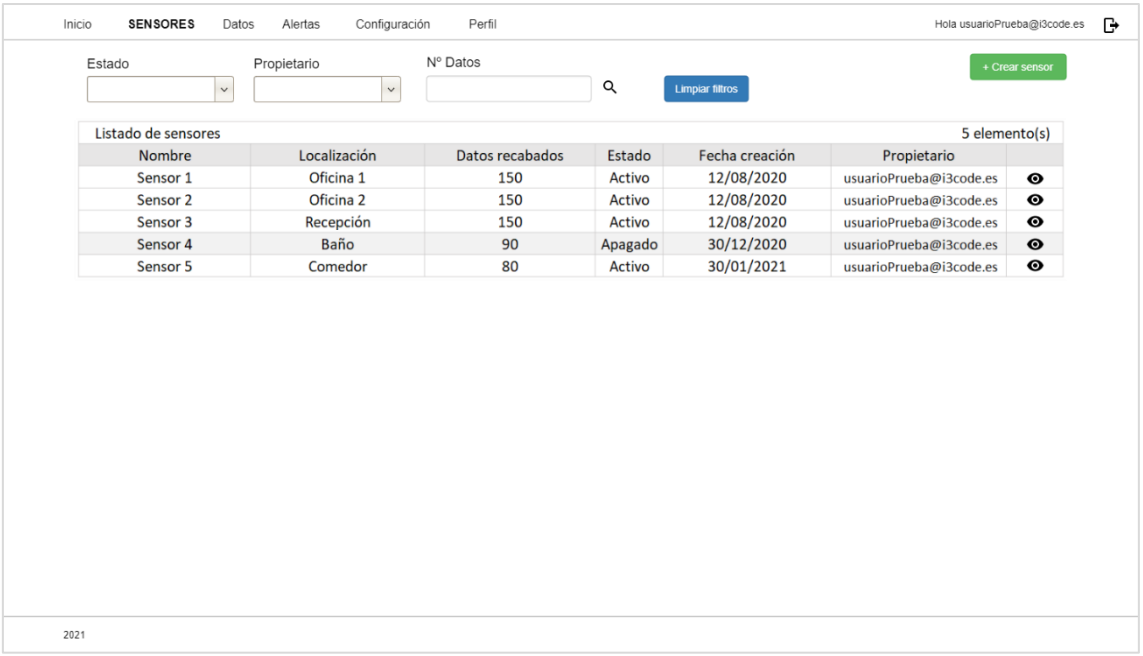

Ilustración 53: Diseño de pantalla de sensores

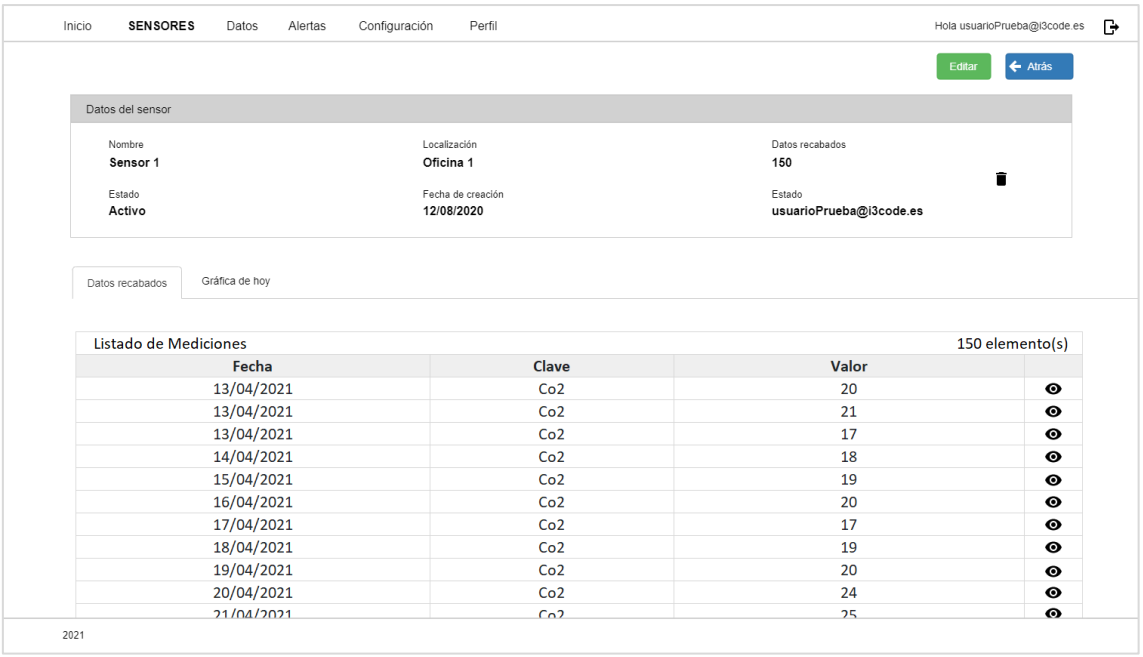

Ilustración 54: Diseño de pantalla de vista de sensor (tabla)

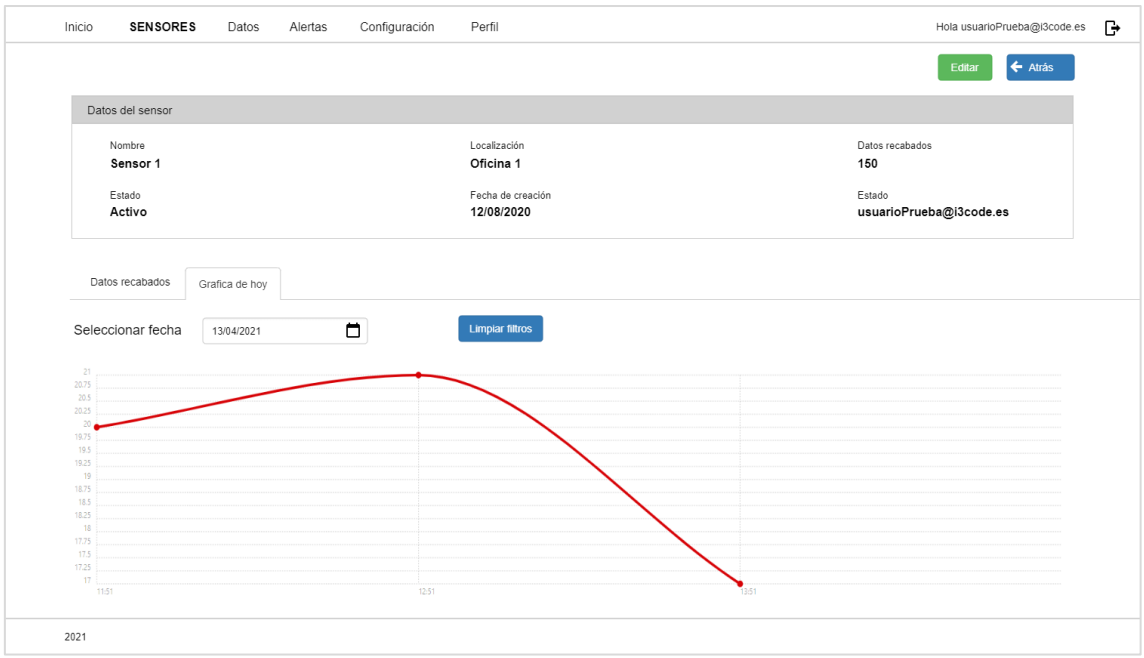

Ilustración 55: Diseño de pantalla de vista de sensor (grafico)

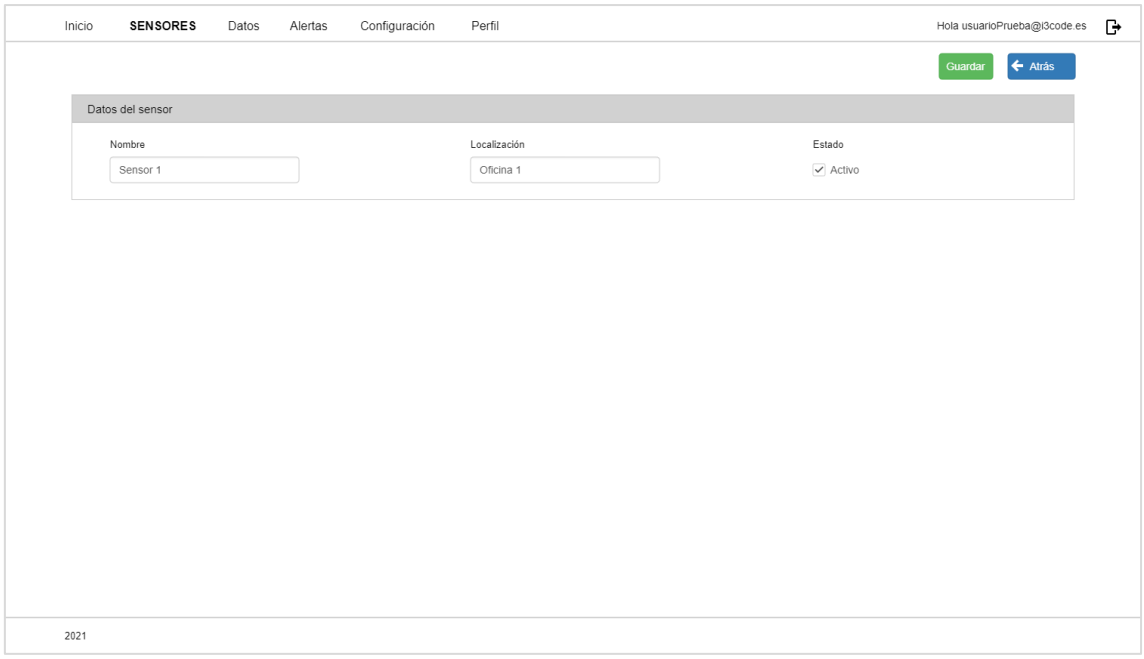

Ilustración 56: Diseño de pantalla de edición del sensor

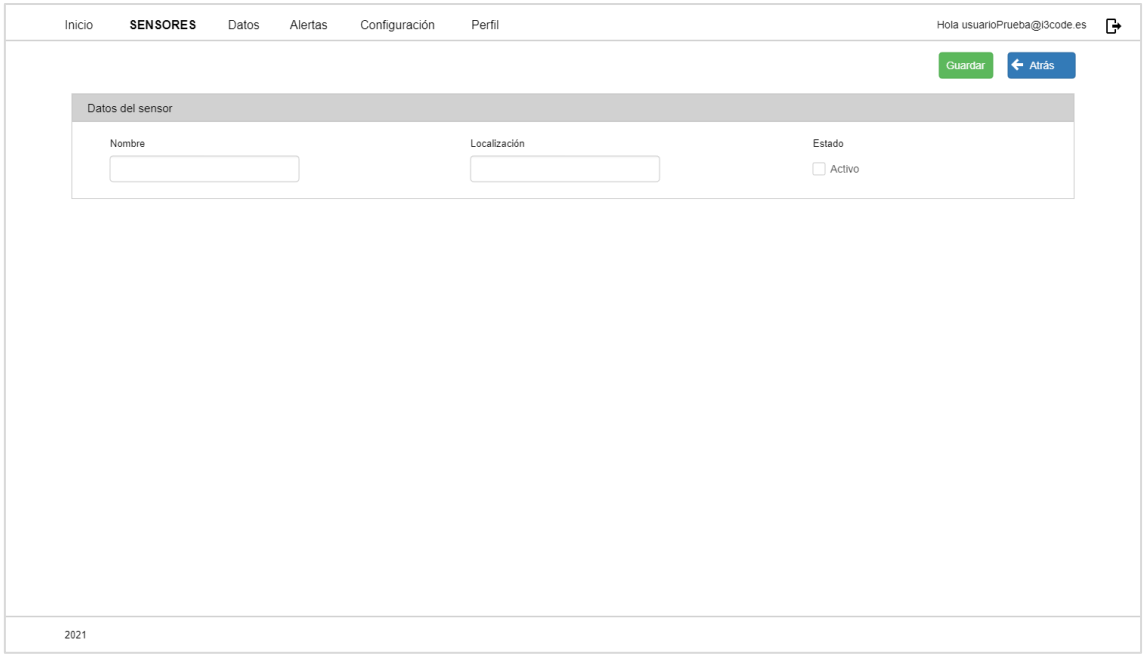

Ilustración 57: Diseño de pantalla de creación del sensor

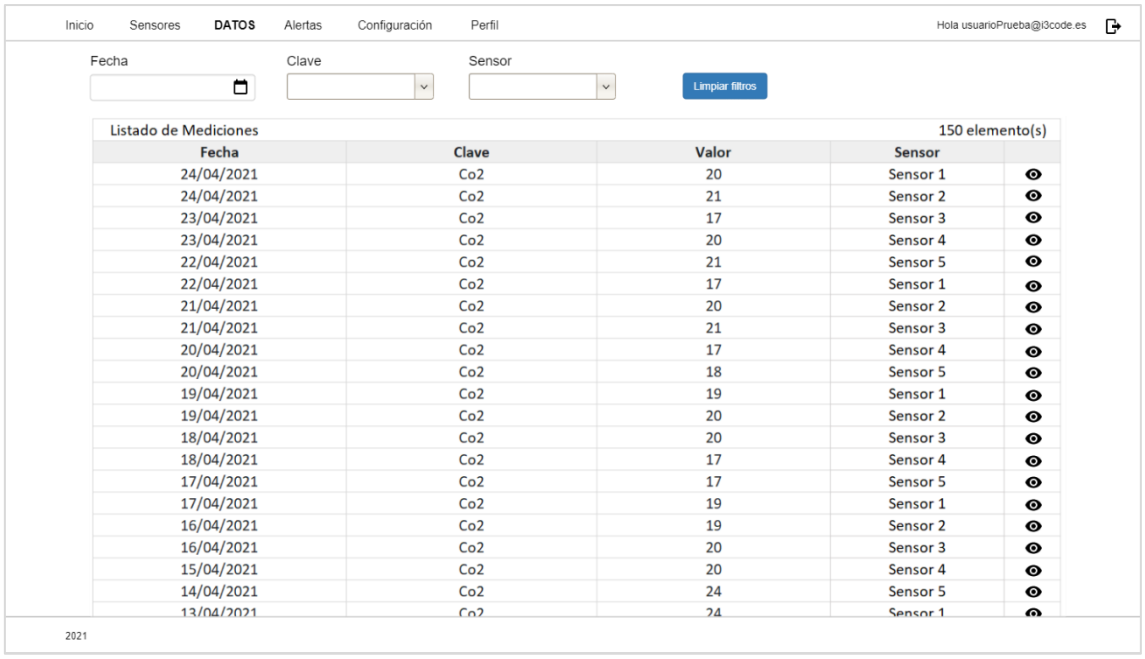

Ilustración 58: Diseño de pantalla de datos

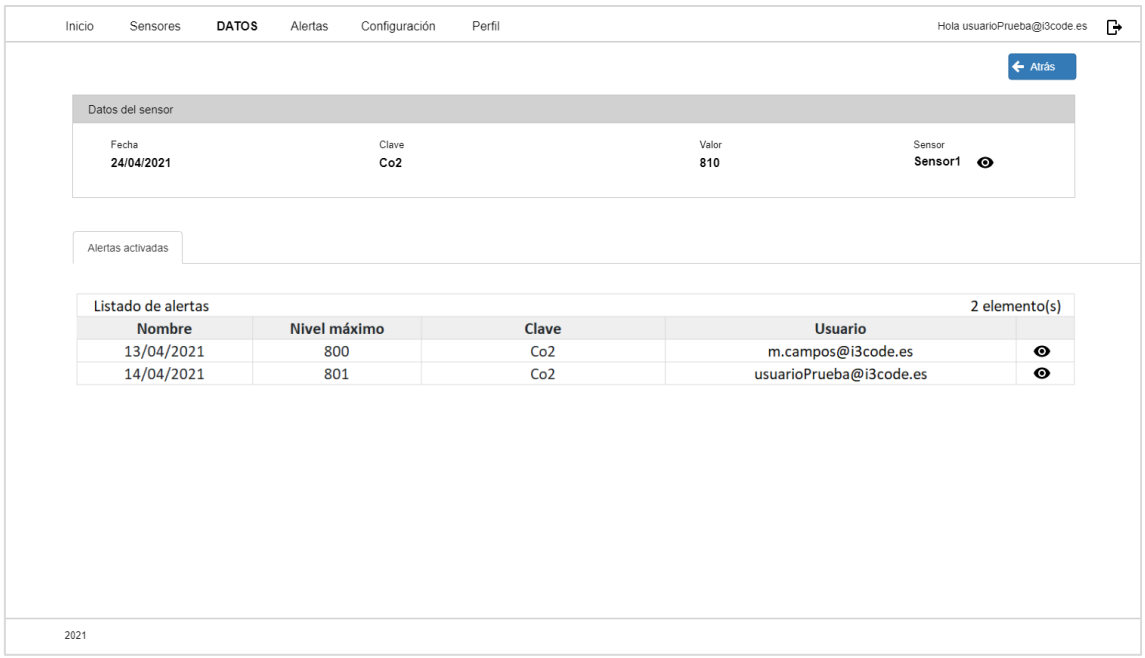

Ilustración 59: Diseño de pantalla de vista del dato

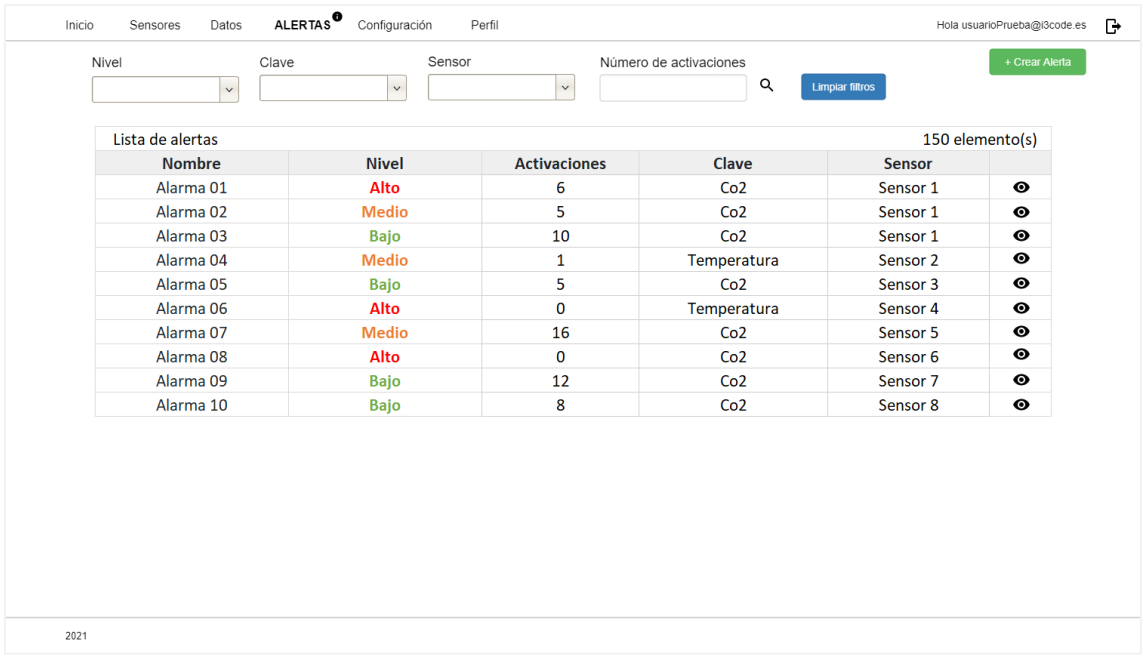

Ilustración 60: Diseño de pantalla alertas

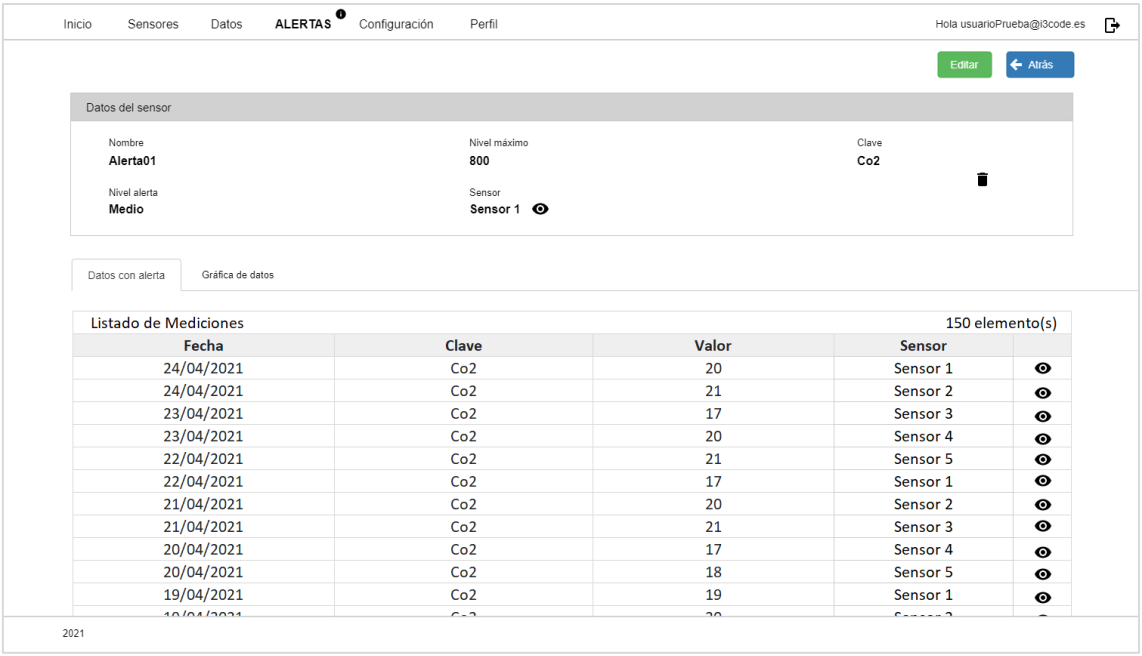

Ilustración 61: Diseño de pantalla de vista de alerta (tabla)

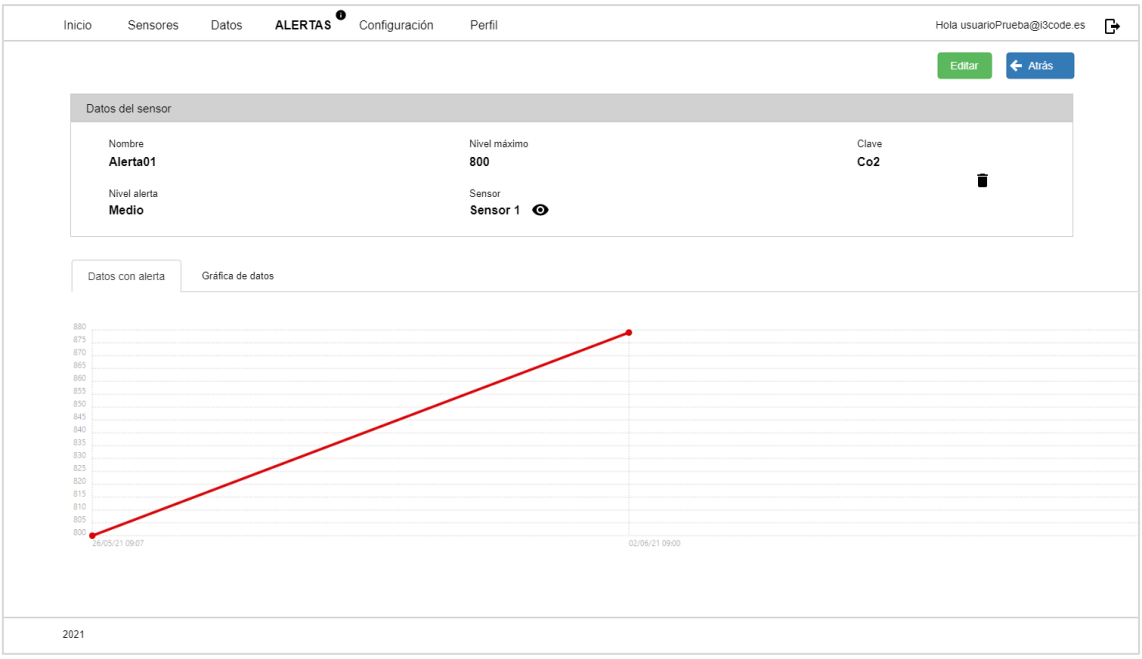

Ilustración 62: Diseño de pantalla de vista de alerta (grafico)

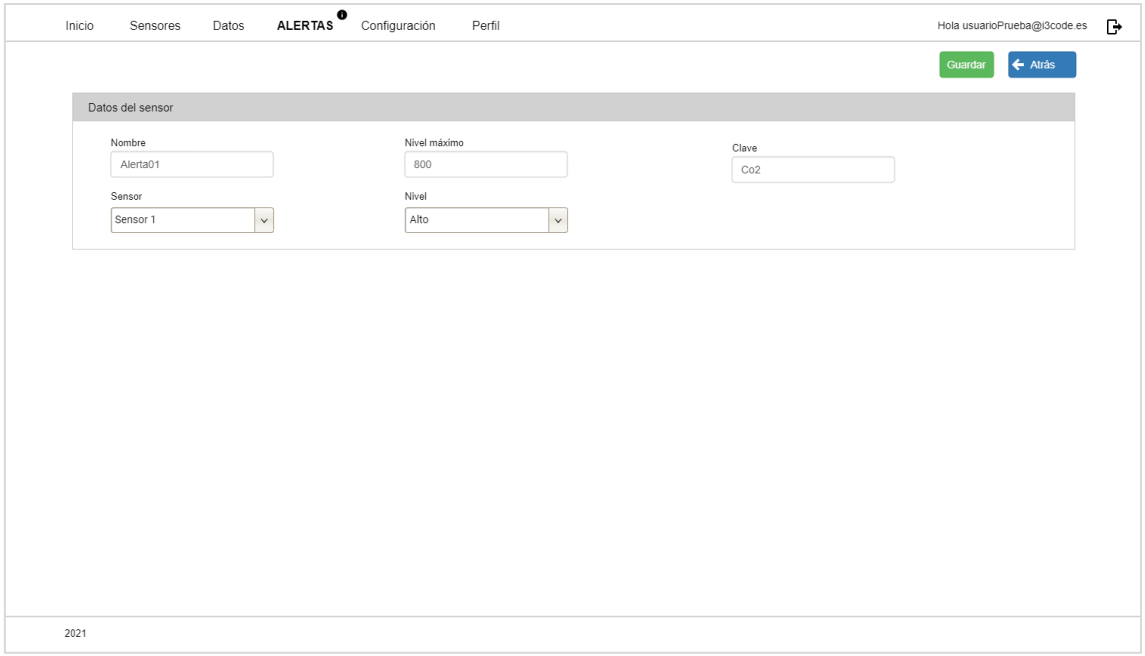

Ilustración 63: Diseño de pantalla de edición de alerta

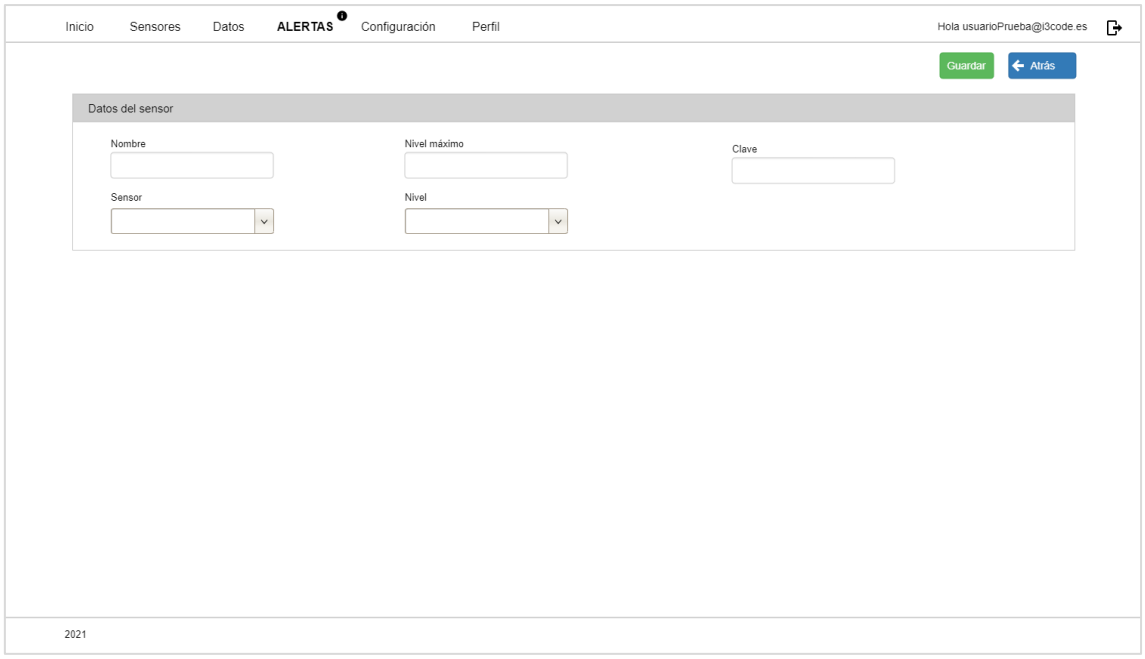

Ilustración 64: Diseño de pantalla de creación de alerta

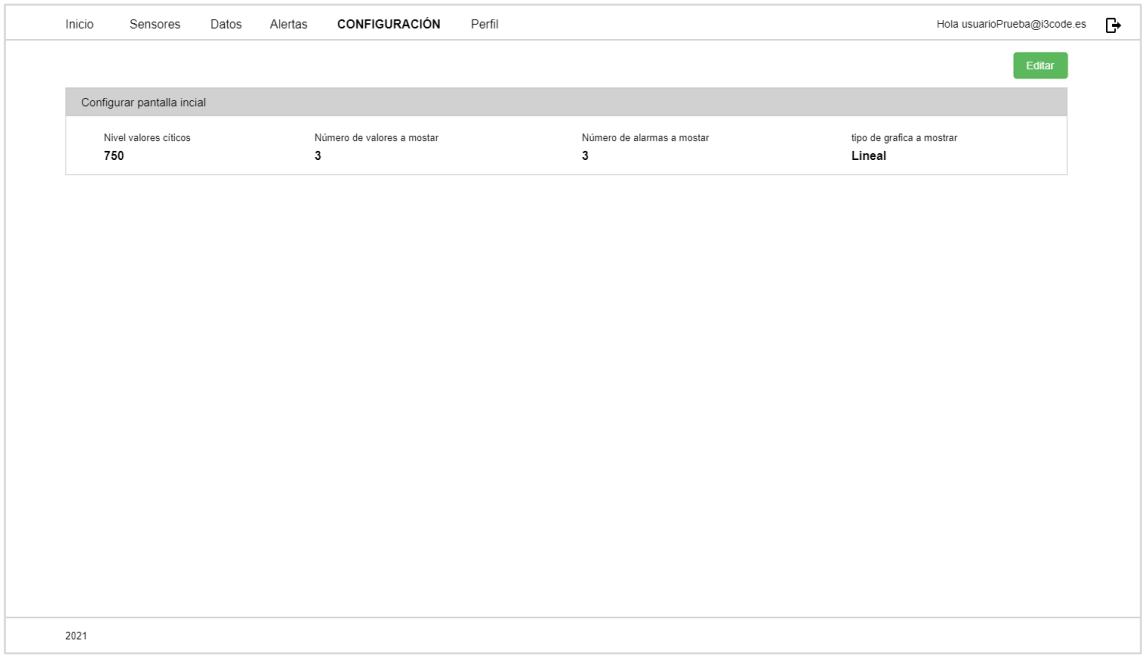

Ilustración 65: Diseño de pantalla de configuración

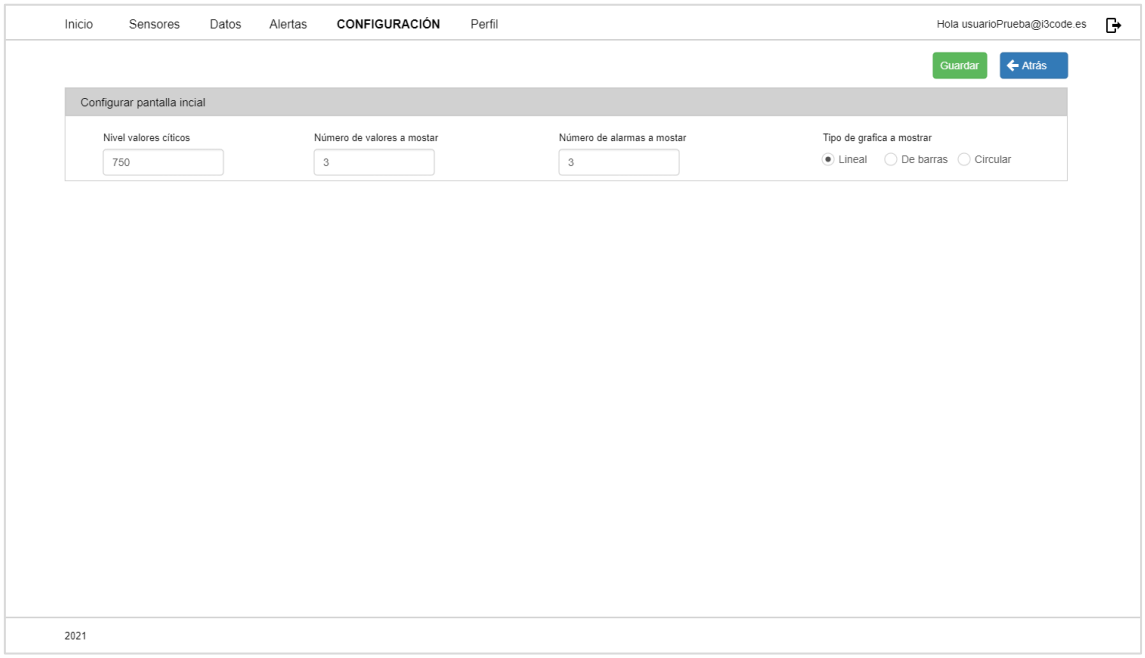

Ilustración 66: Diseño de pantalla de edición de configuración

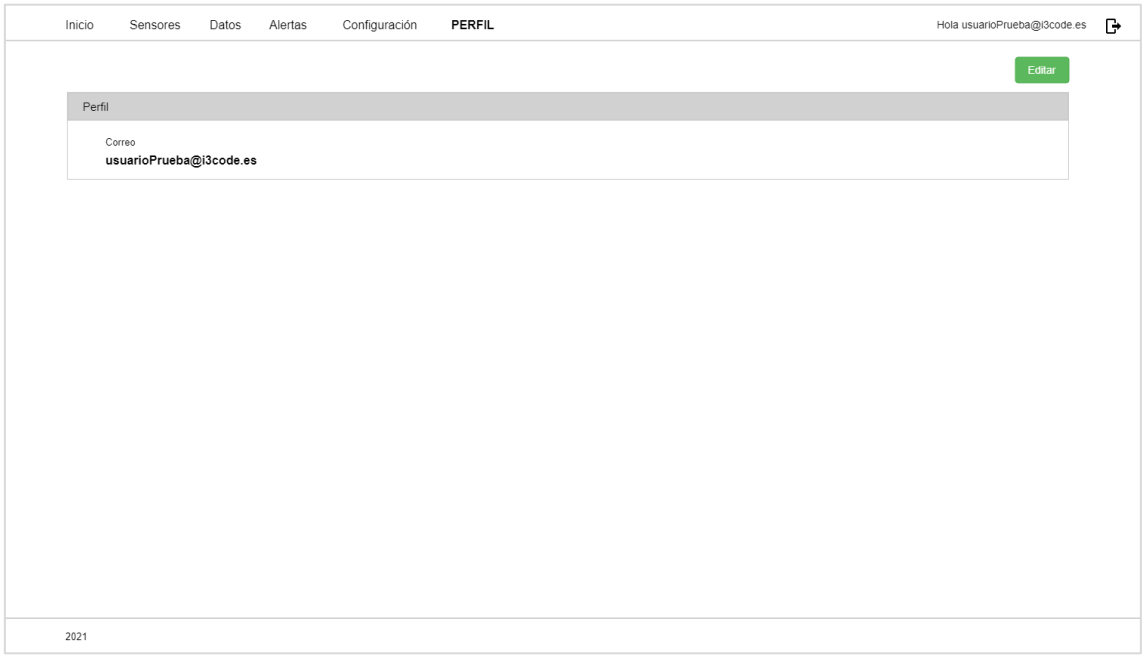

Ilustración 67: Diseño de pantalla de perfil

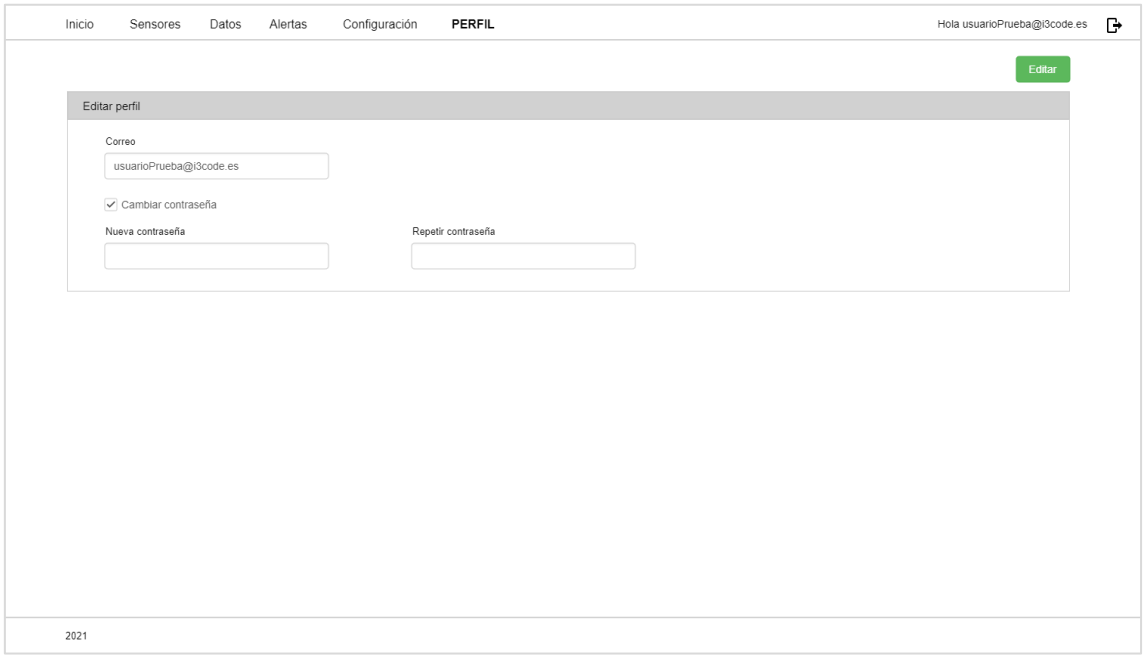

Ilustración 68: Diseño de pantalla de edición de perfil

Anexo 11 - Diseño de base de datos obtenido en la etapa de diseño

La parte de la base de datos ha sido totalmente dirigida por la empresa i3Code ya que se pretende que sea lo más parecida a la que disponen sus clientes para así poder integrarlo de forma sencilla con las aplicaciones web que i3Code ya ha desarrollado para ellos. Por tanto, en cuanto a la base de datos solo se ha tenido en cuenta su diseño en cuanto al almacenamiento de los datos.

Modelo entidad-relación

Para este diseño se ha comenzado por el modelo entidad-relación. Se trata de un modelo que permite representar los componentes que participan en un proceso y como se relacionan entre sí. El modelo entidad relación cuenta con:

- Entidades: representan personas, cosas u objetos diferenciables entre sí. Estas entidades son cada uno de los componentes que participan en el proceso.
- Relaciones: representan, como su nombre indica, las relaciones entre las entidades. Estas relaciones expresan como se afectan los componentes del proceso.
- Cardinalidad: tipo de relación entre entidades.
    - Uno a uno: un individuo de la entidad primera solo puede relacionarse con un individuo de la entidad segunda y viceversa.
    - Uno a varios: un individuo de la entidad primera puede relacionarse con uno o varios individuos de la entidad segunda, pero los individuos de la entidad segunda solo pueden estar relacionados con uno de la entidad primera.
    - Varios a varios: un individuo de la entidad primera puede relacionarse con uno o varios individuos de la entidad segunda y viceversa.

Las entidades se representan en forma de rectángulos, las relaciones en forma de rombo y las cardinalidades de forma numérica sobre las líneas que unen las entidades y las relaciones. En el caso del modelo entidad-relación de este proyecto, se pueden encontrar las entidades *Empresa*, *Sensor*, *Usuario*, *Telemetría*, *Alerta* y *Configuración*. También relaciones como *pertenece* y *tiene*. Y por último cardinalidades de tipo uno a uno en la relación entre el *Usuario* y la *Configuración*, cardinalidad de tipo muchos a muchos en la relación entre la *Telemetría* y la *Alerta* y cardinalidades de tipo uno a varios en el resto de las relaciones. Por tanto, el modelo entidad-relación ha quedado de la siguiente forma:

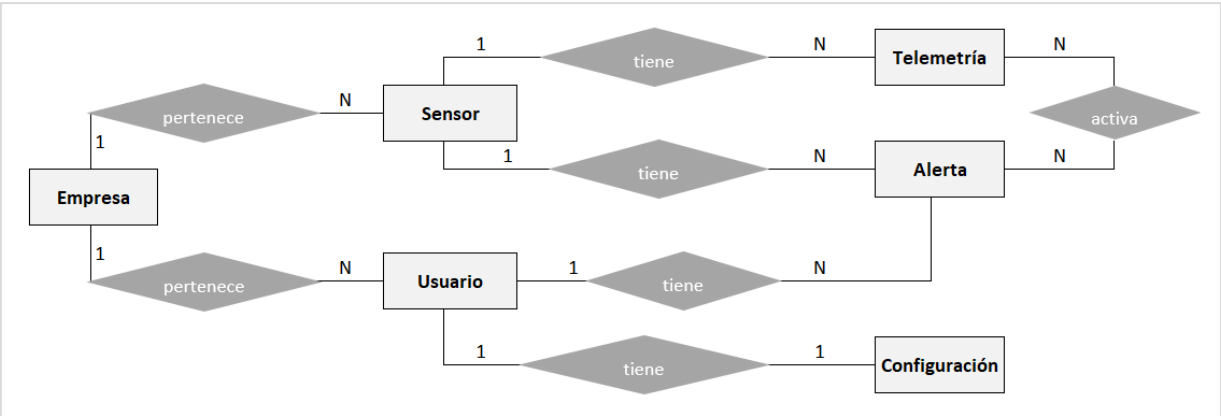

Ilustración 69: Diagrama entidad-relación de la base de datos

Modelo relacional

El modelo relacional consiste en representar datos por medio de tablas. Estas tablas o relaciones están compuestas por tuplas o filas y campos o columnas. Estas tablas almacenan los registros, que son la información de un individuo concreto. Cada tabla cuenta con un campo denominado clave primaria que representa el campo dentro del registro de cada individuo que lo identifica de forma única. Además, existen las claves foráneas que son campos de una tabla que identifican a otra porque son su clave primaria, de esta forma se crean las relaciones entre tablas. En muchas ocasiones, este modelo se obtiene de convertir el modelo entidad-relación en el modelo relacional por lo que las relaciones del primero deben desaparecer convirtiéndose en las claves foráneas. La forma más lógica de eliminar una relación es la de añadir la clave primaria de una de las entidades en la otra participe en la relación, pero el problema viene cuando aparecen cardinalidades de varios, ya sea de uno a varios o de varios a varios. En el primero de los casos la solución es sencilla, ya que una de las entidades solo se puede relacionar con uno de los individuos de la otra, por lo que es esta entidad la que obtendrá la clave primaria de la otra. En el caso de tener cardinalidad varios a varios la solución más sencilla y eficientes es la de crear una tabla auxiliar que contenga ambas claves primarias y en ocasiones algún campo que haga referencia a esa relación.

En el caso del modelo relacional del proyecto actual, se cuentan con las tablas: *Empresa*, *Usuario*, *Configuración*, *Sensor*, *Telemetría*, *Alerta* y *Alerta_log*. En el caso de la relación uno a uno entre el *Usuario* y la *Configuración* se ha decidido añadir la clave primaria del *Usuario* en la tabla de la *Configuración* porque, aunque no haya diferencias, se ha comprendido que la configuración depende del usuario. En caso de las relaciones de uno a muchos se ha añadido la clave primaria como se ha explicado anteriormente. Por último, en el caso de la relación muchos a muchos entre *Telemetría* y *Alerta* se ha realizado una tabla auxiliar que simplemente contiene su propia clave primaria y las claves de las tablas relacionadas. El modelo ha quedado de la siguiente forma:

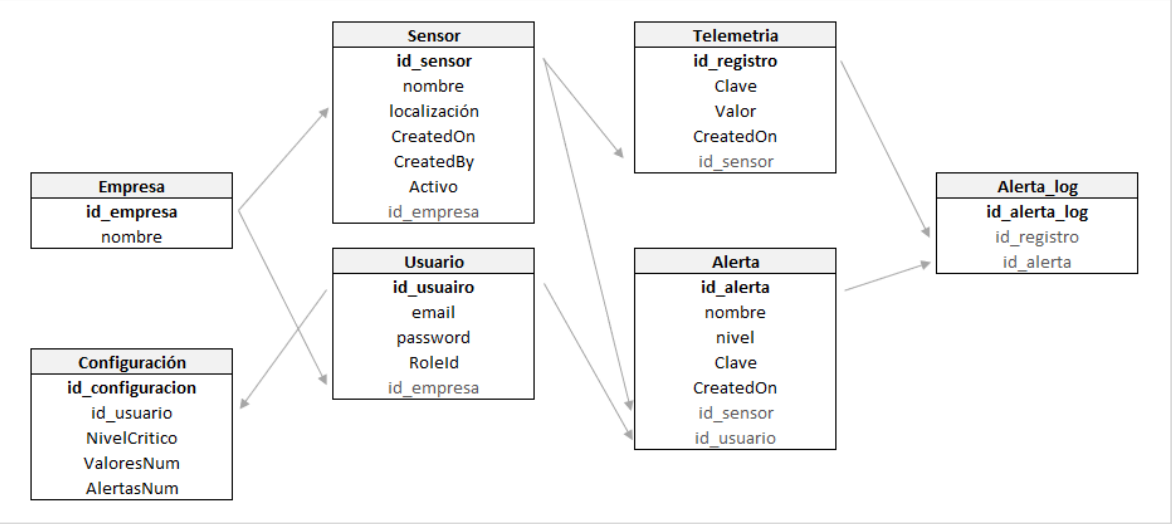

Ilustración 70: Diagrama del modelo relacional de la base de datos

Anexo 12 - Capturas de la aplicación web desarrollada

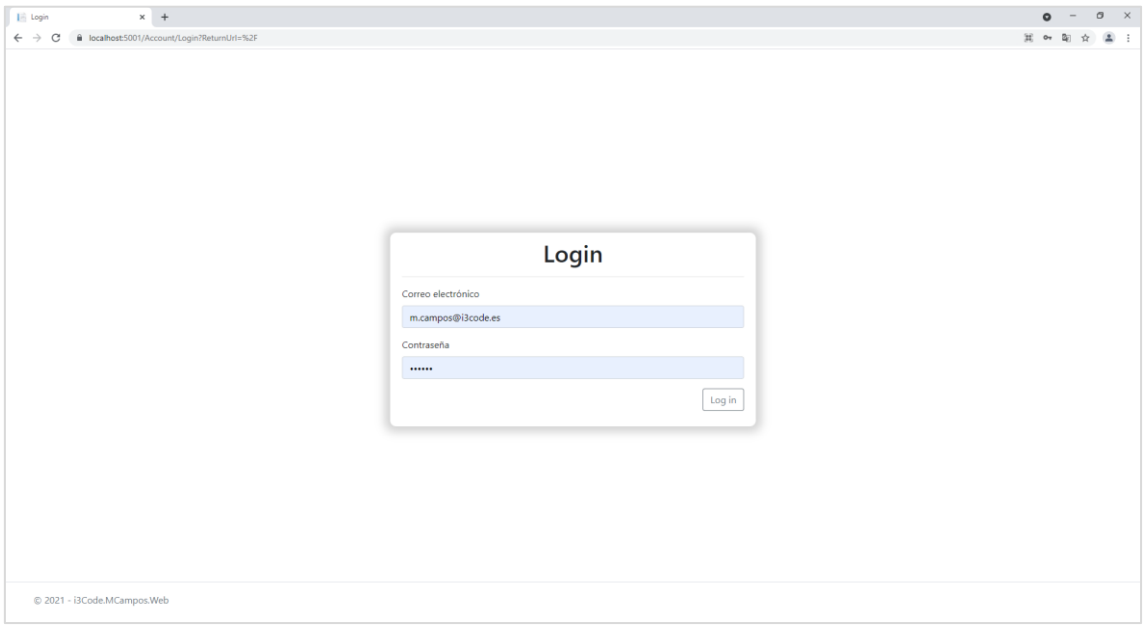

Ilustración 71: Captura de la pantalla de inicio de sesión o login

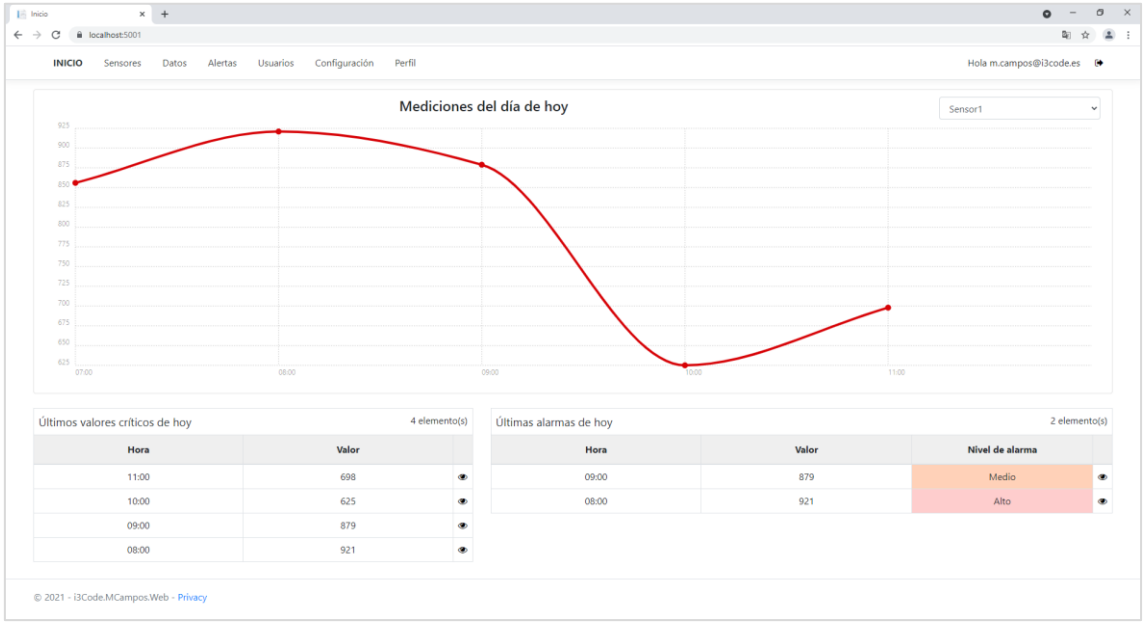

Ilustración 72: Captura de la pantalla de inicio (grafico lineal)

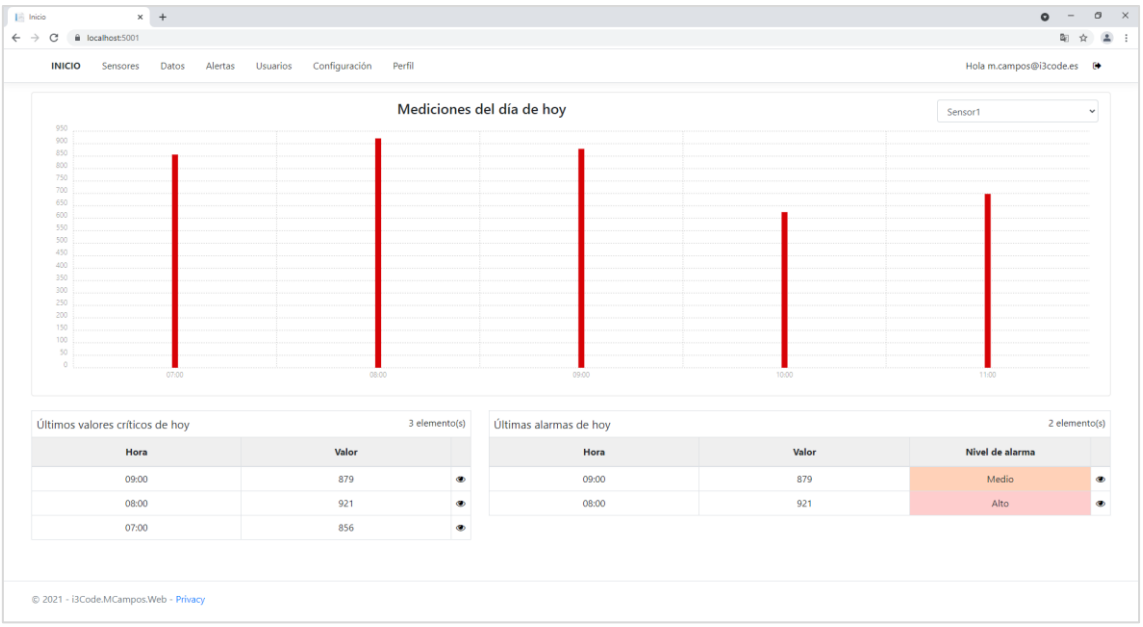

Ilustración 73: Captura de la pantalla de inicio (grafico de barras)

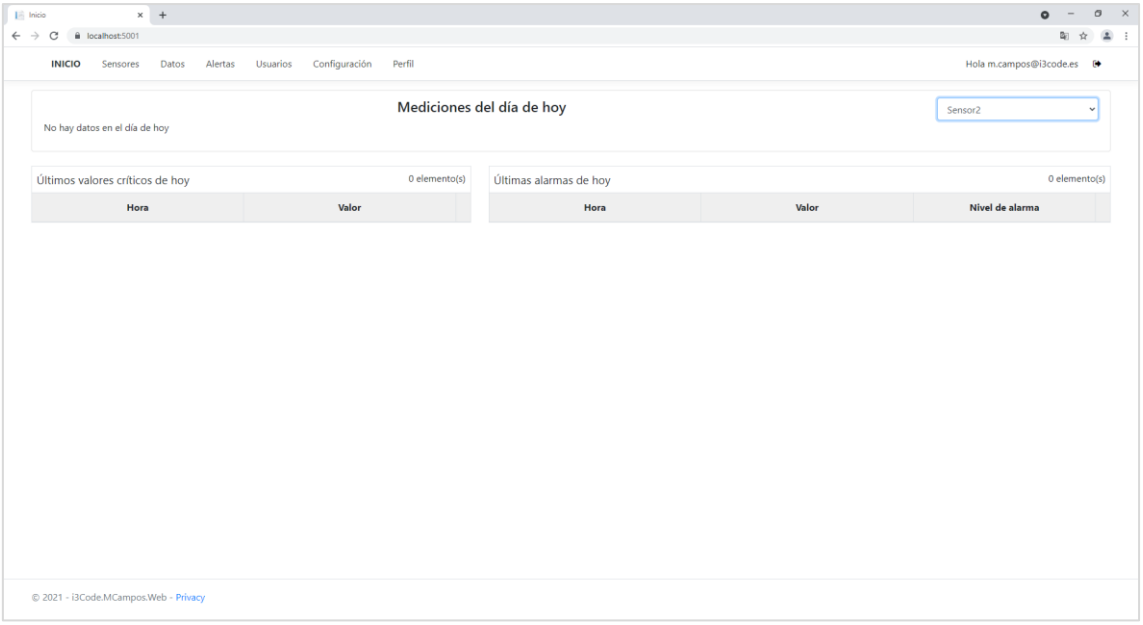

Ilustración 74: Captura de la pantalla de inicio (son datos)

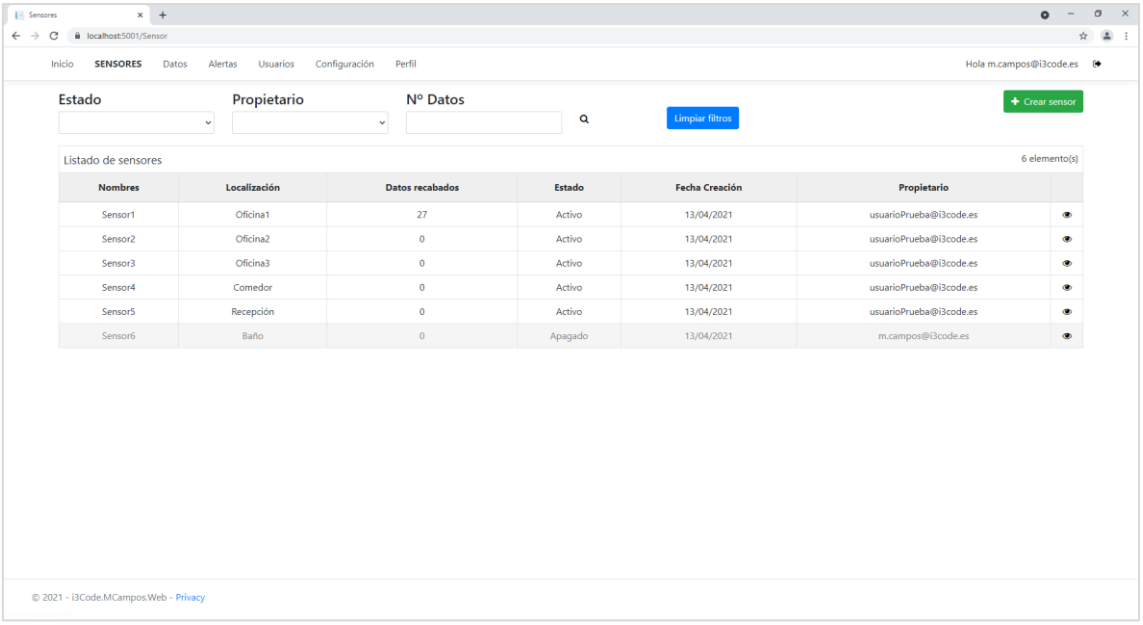

Ilustración 75: Captura de la pantalla de sensores

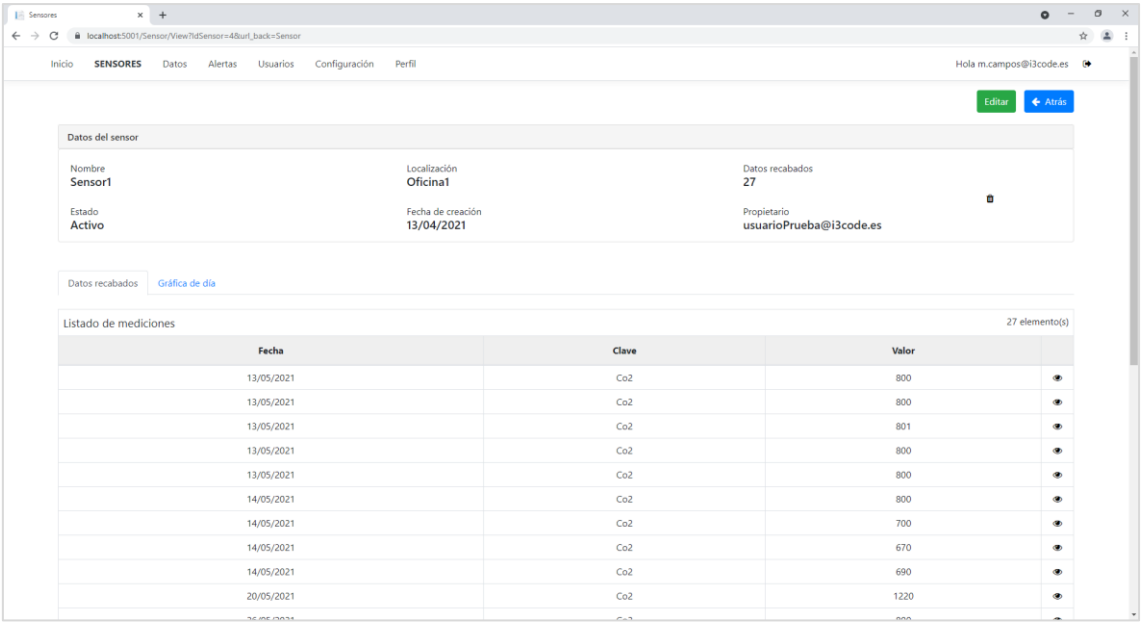

Ilustración 76: Captura de la pantalla de vista de sensor (tabla)

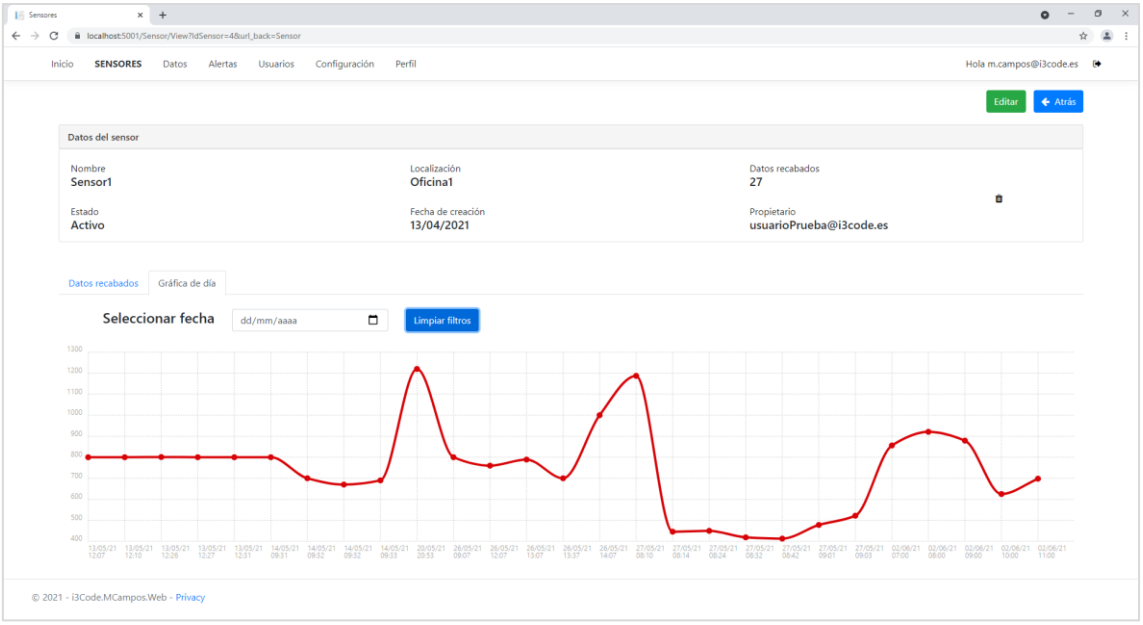

Ilustración 77: Captura de la pantalla de vista de sensor (grafico)

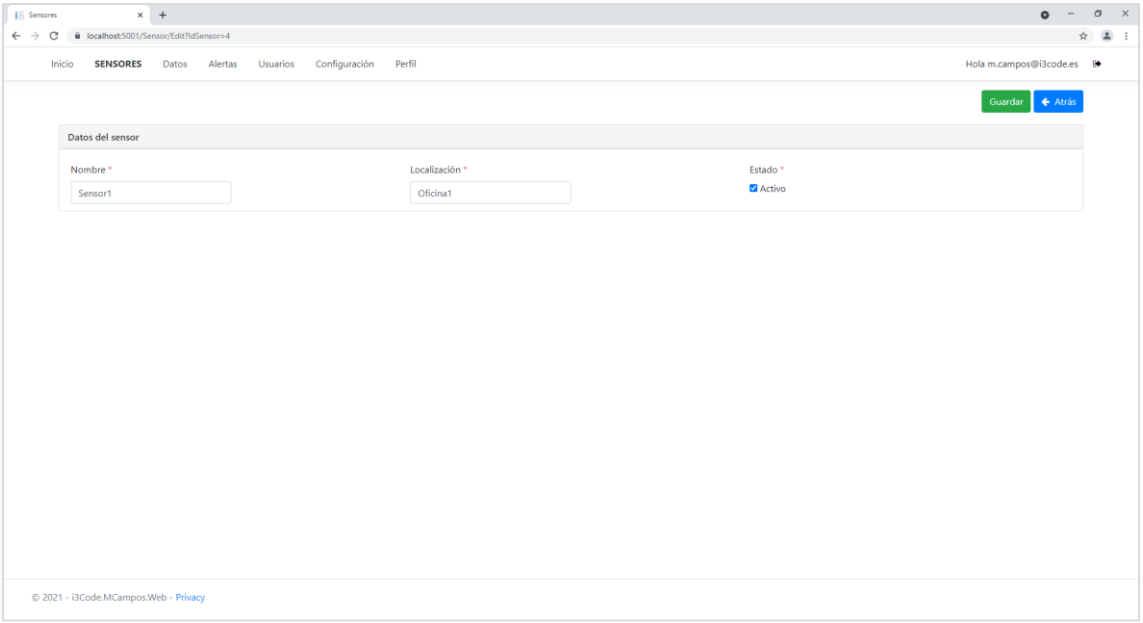

Ilustración 78: Captura de la pantalla de edición de sensor

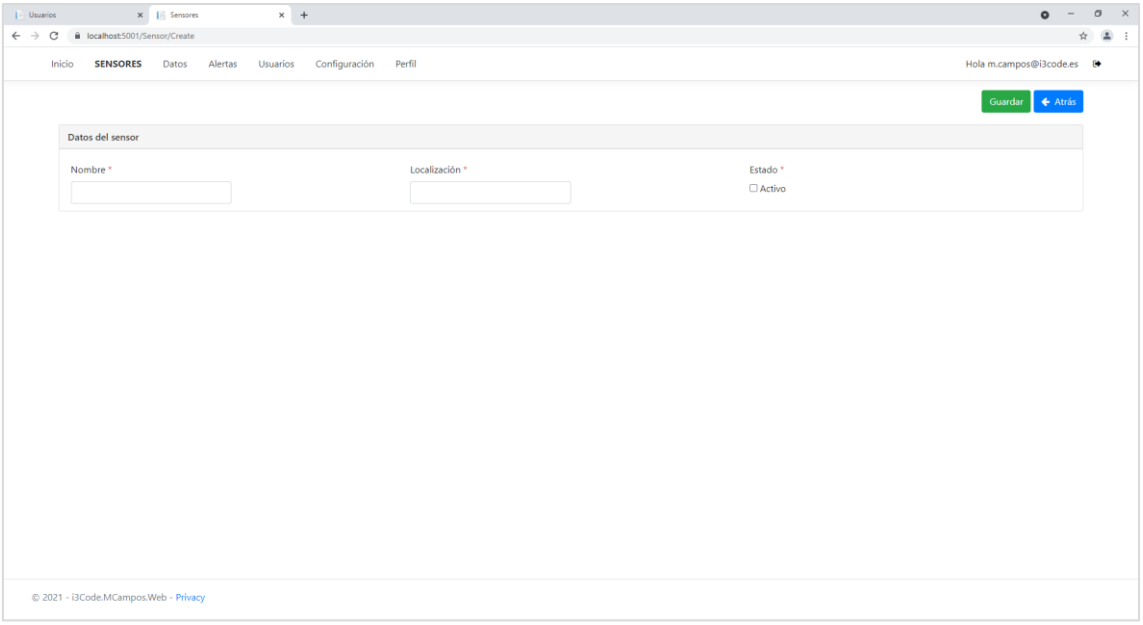

Ilustración 79: Captura de la pantalla de creación de sensor

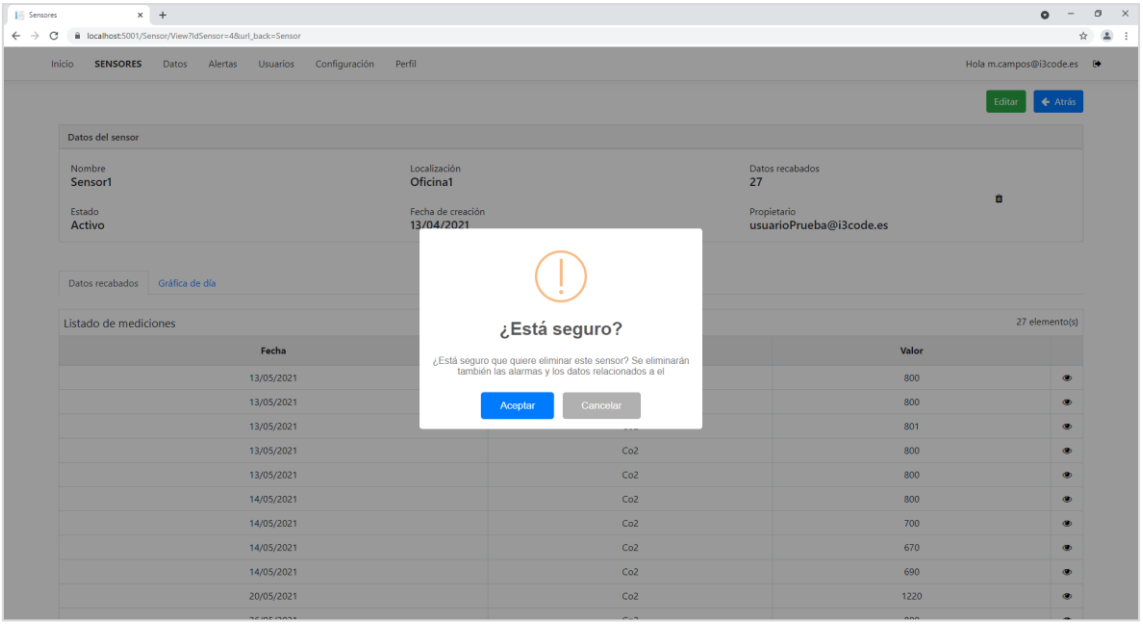

Ilustración 80: Captura de la pantalla de ejemplo de eliminación de un elemento

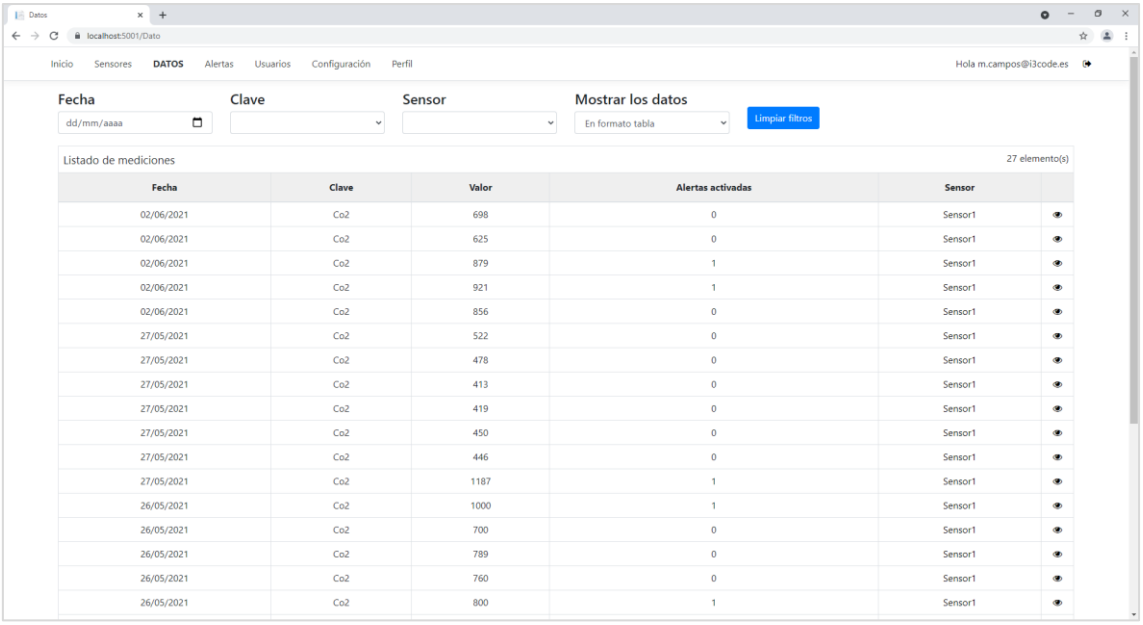

Ilustración 81: Captura de la pantalla de datos (tabla)

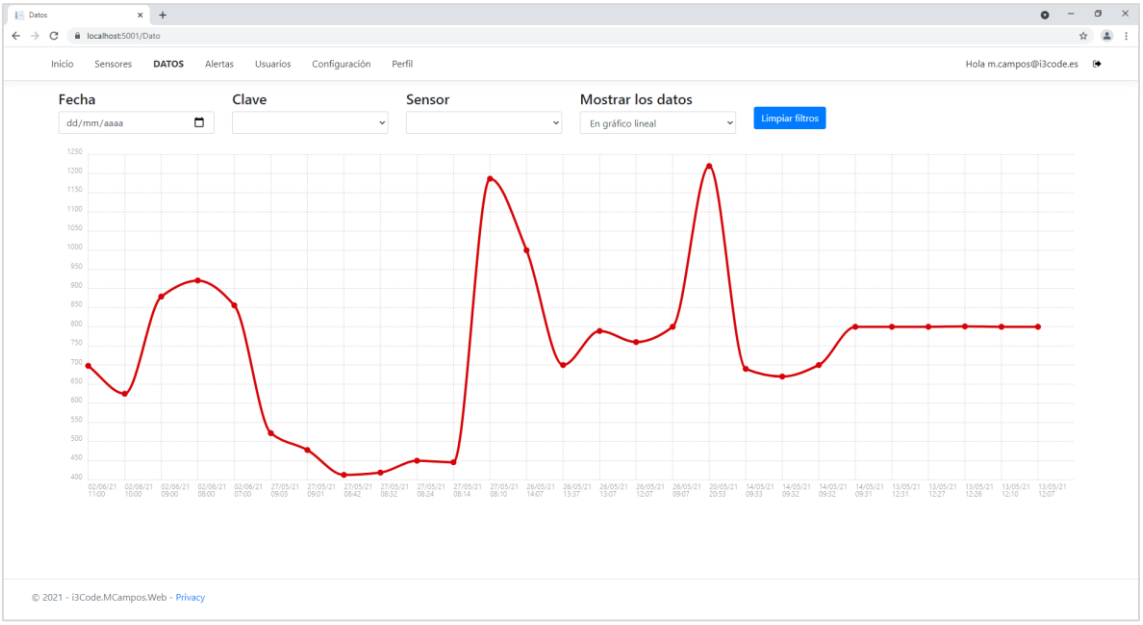

Ilustración 82: Captura de la pantalla de datos (grafico lineal)

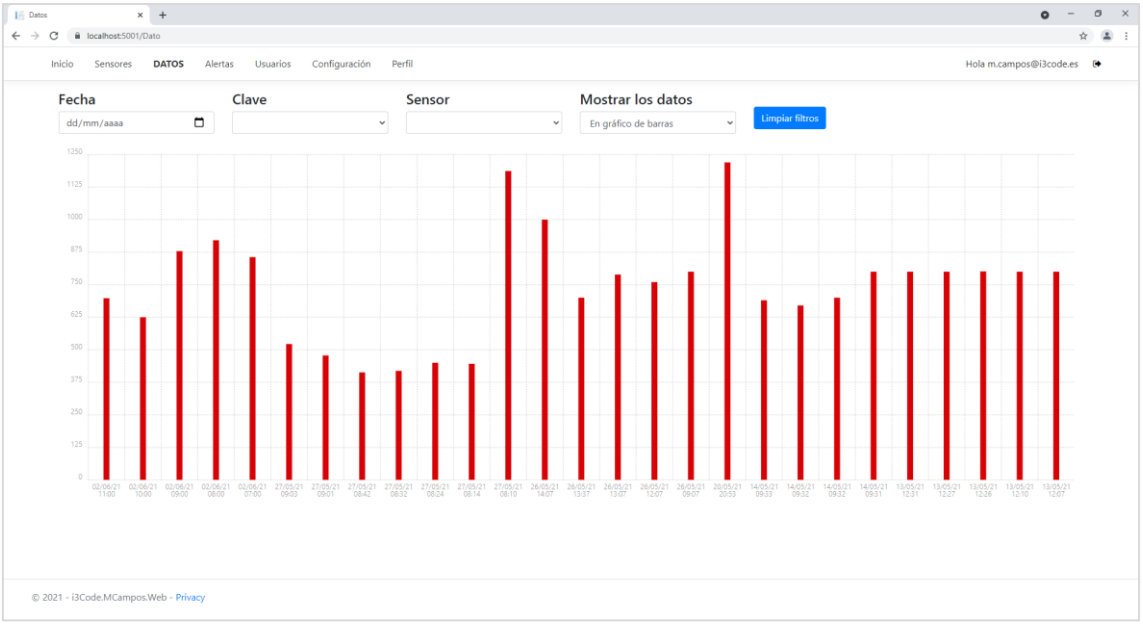

Ilustración 83: Captura de la pantalla de datos (grafico de barras)

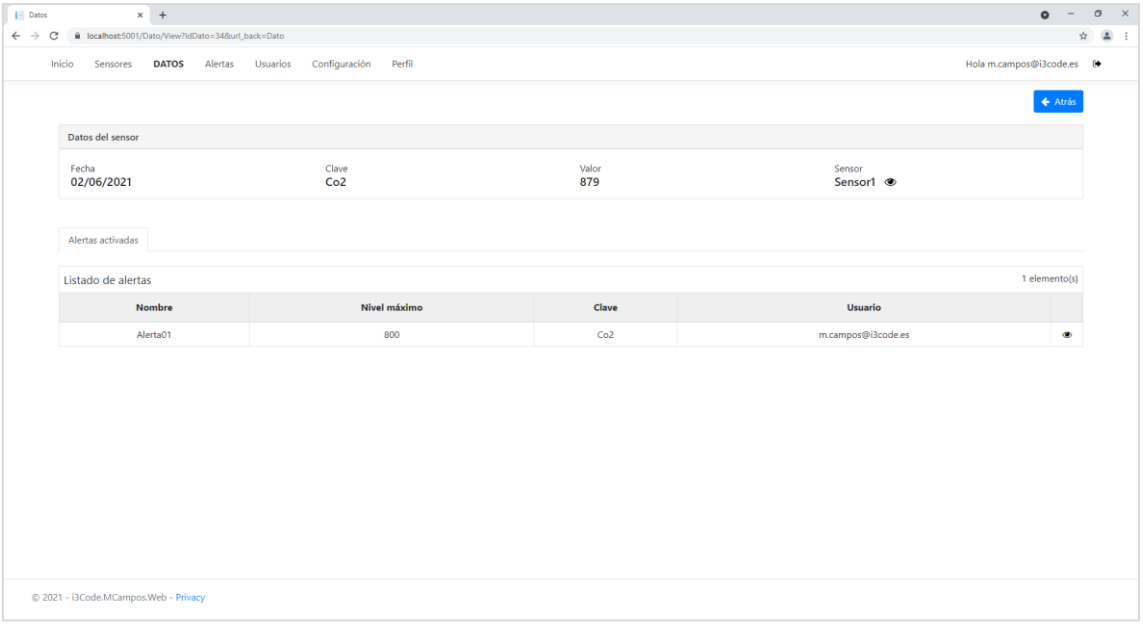

Ilustración 84: Captura de la pantalla de vista de dato

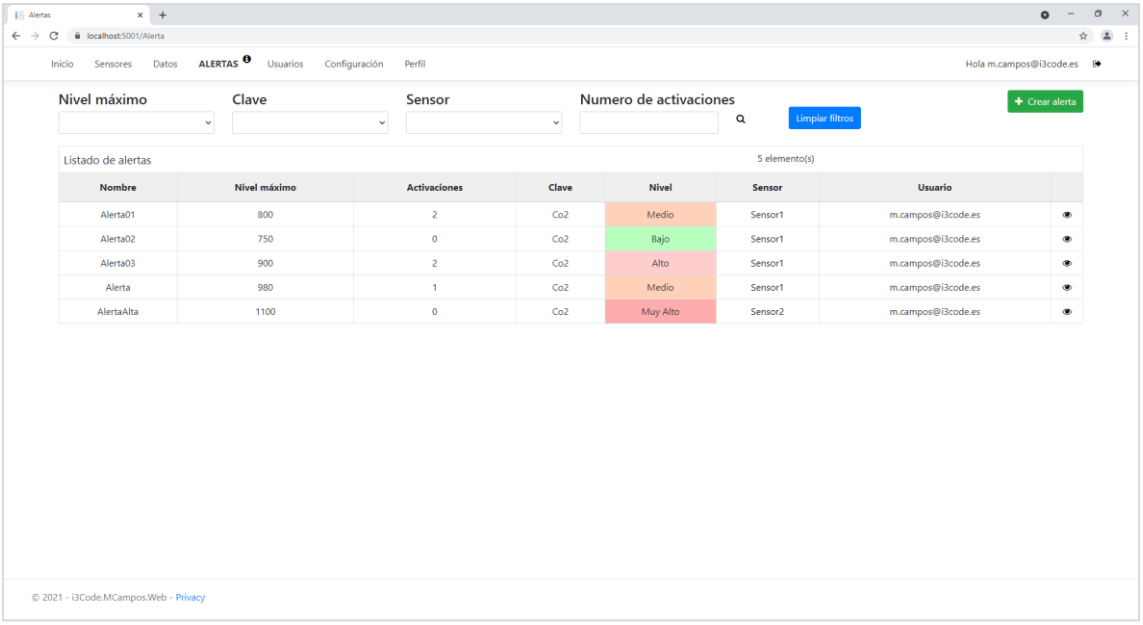

Ilustración 85: Captura de la pantalla de alertas

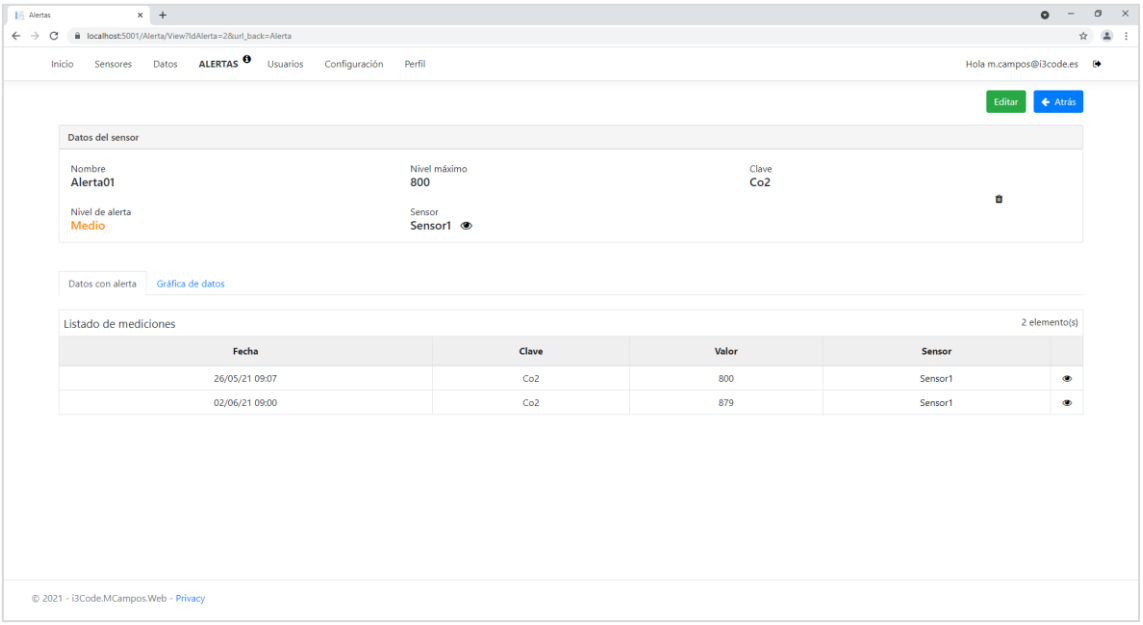

Ilustración 86: Captura de la pantalla de vista de alerta (tabla)

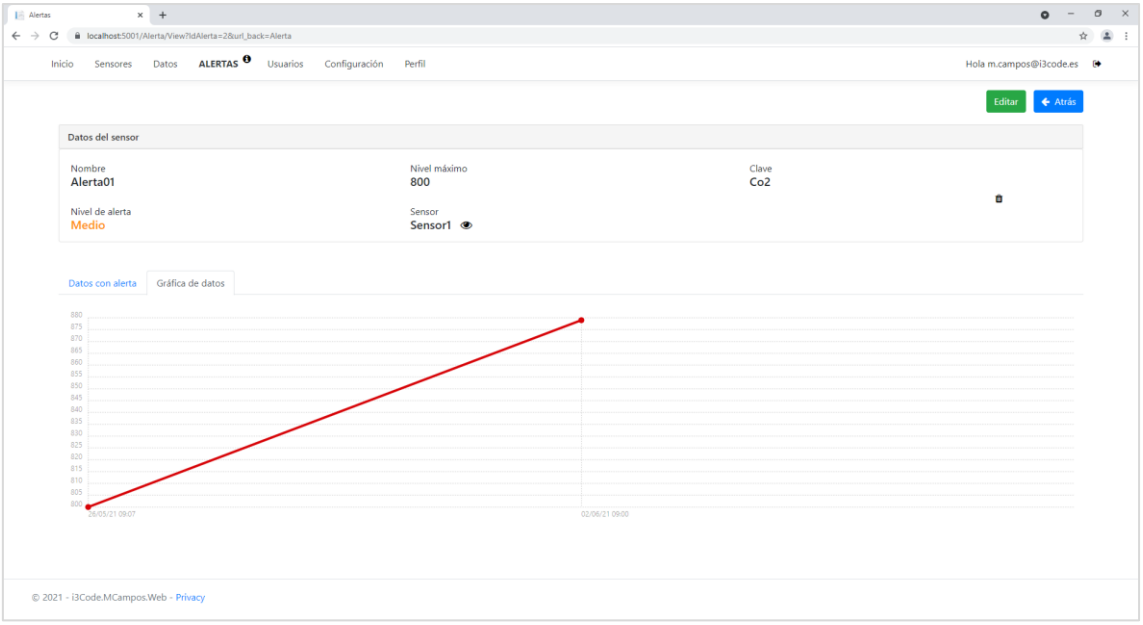

Ilustración 87: Captura de la pantalla de vista de alerta (grafico)

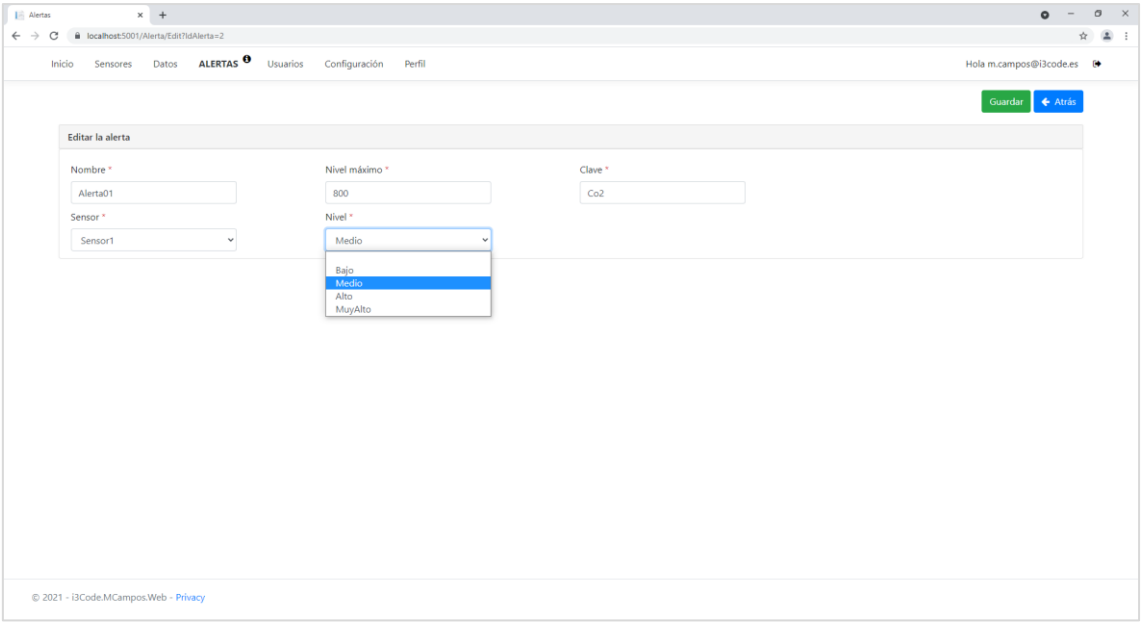

Ilustración 88: Captura de la pantalla de edición de alerta

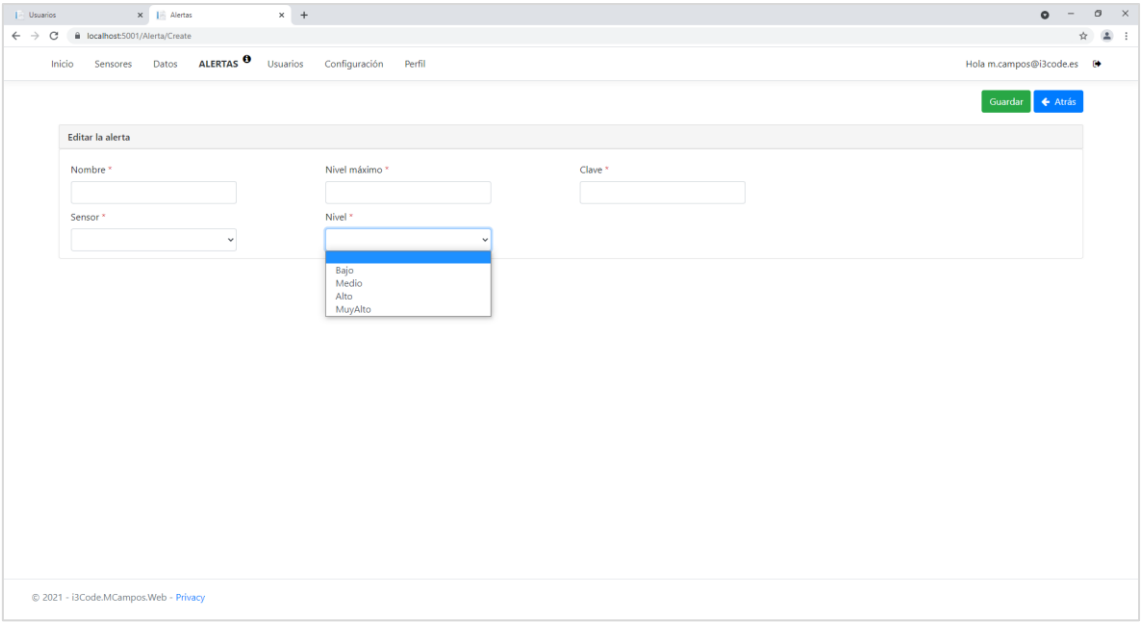

Ilustración 89: Captura de la pantalla de creación de alerta

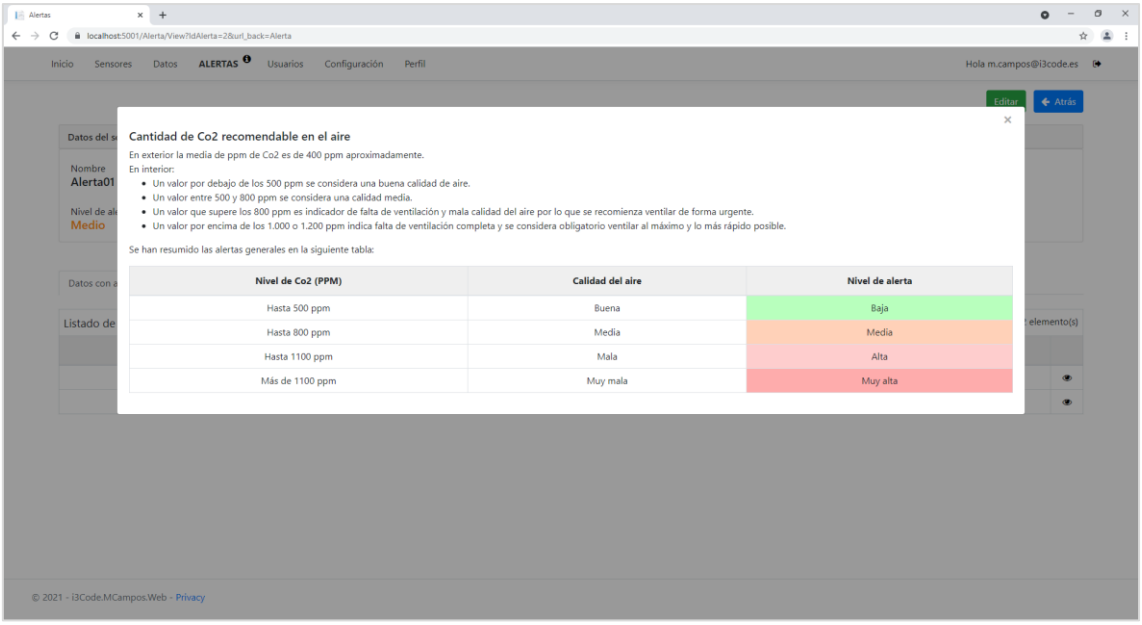

Ilustración 90: Captura de la pantalla de información sobre alertas y calidad de aire

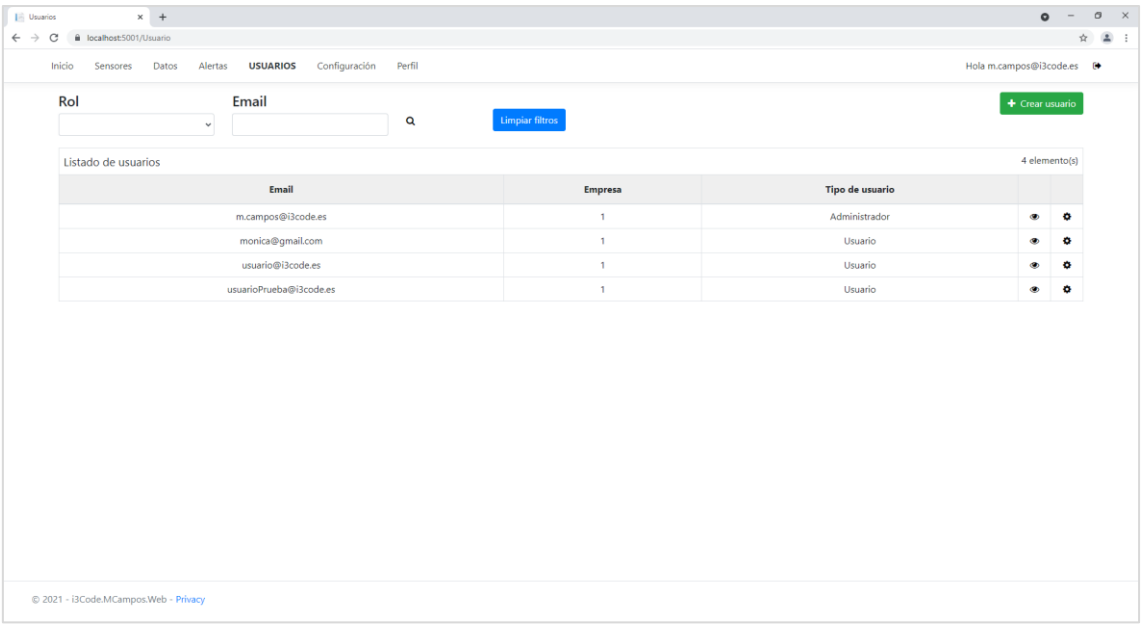

Ilustración 91: Captura de la pantalla de usuarios (solo administrador)

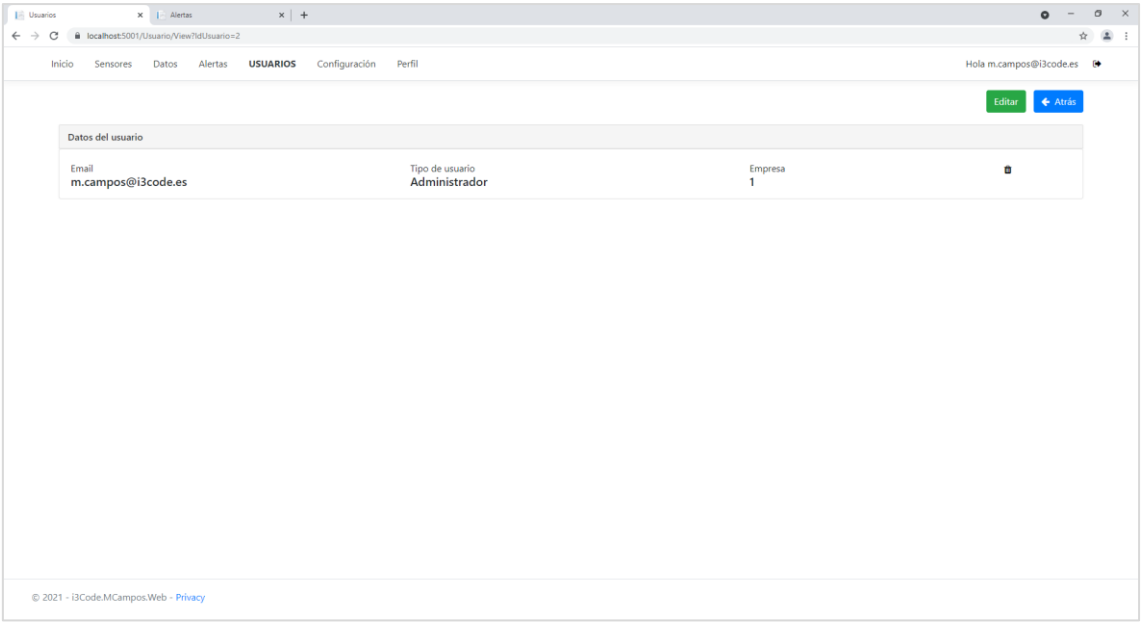

Ilustración 92: Captura de la pantalla de vista de usuario (solo administrador)

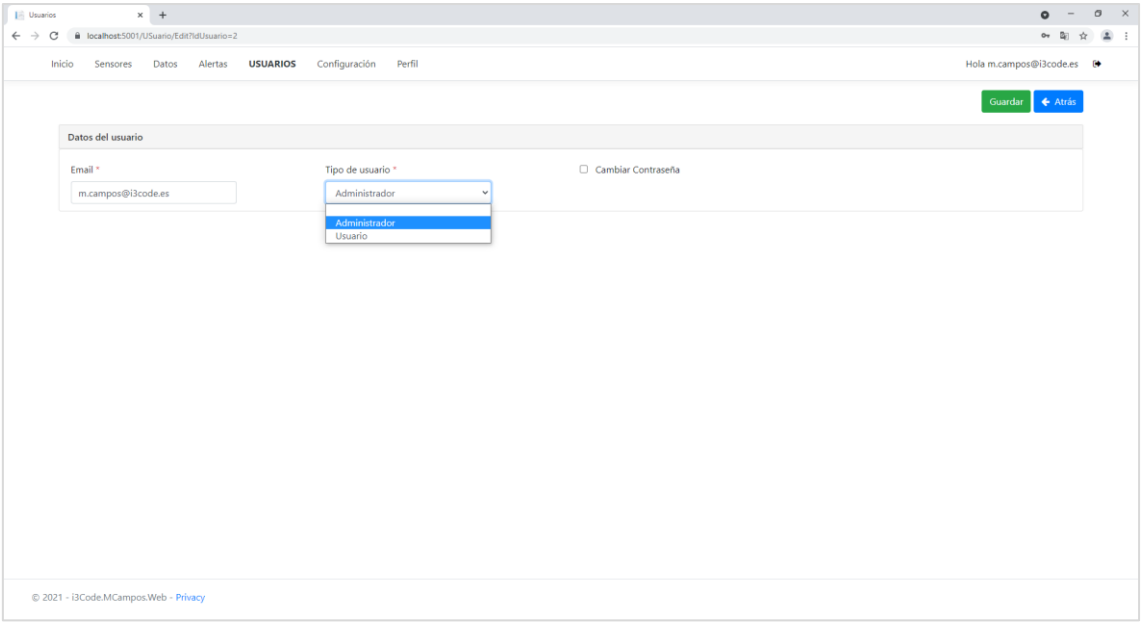

Ilustración 93: Captura de la pantalla de edición del usuario son contraseña (solo administrador)

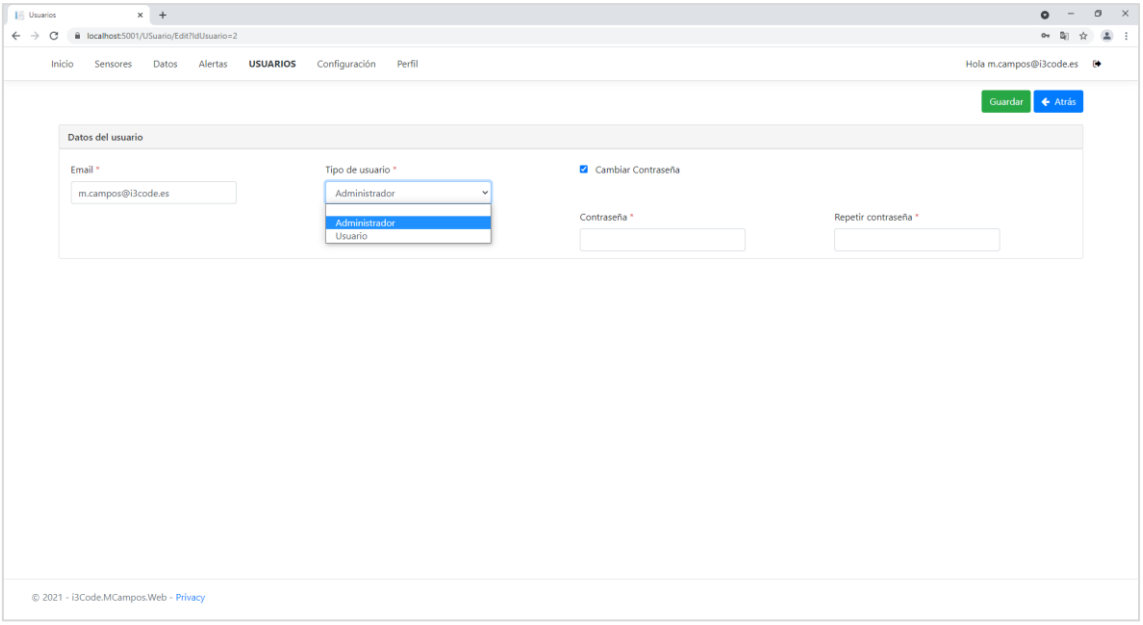

Ilustración 94: Captura de la pantalla de edición del usuario con contraseña (solo administrador)

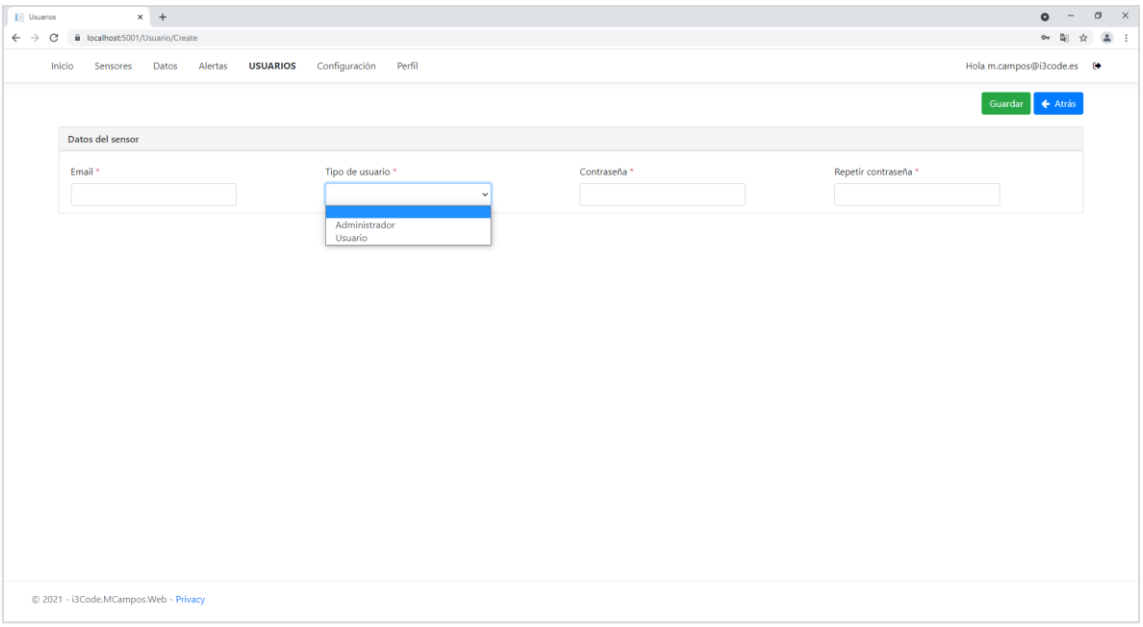

Ilustración 95: Captura de la pantalla de creación del usuario (solo administrador)

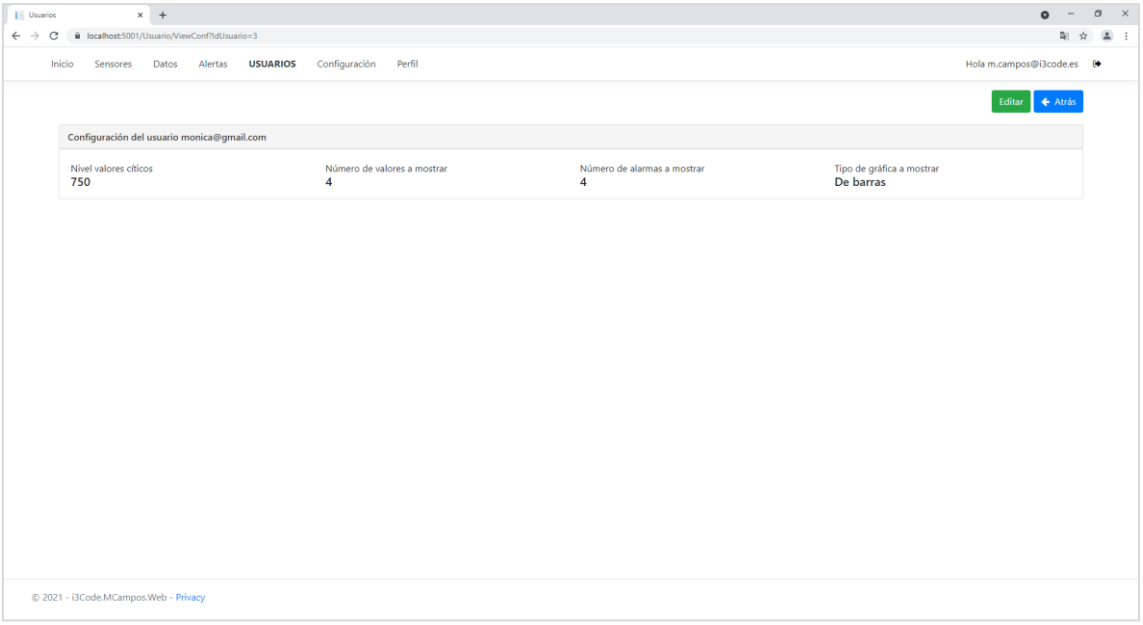

Ilustración 96: Captura de la pantalla de vista de la configuración de usuario (solo administrador)

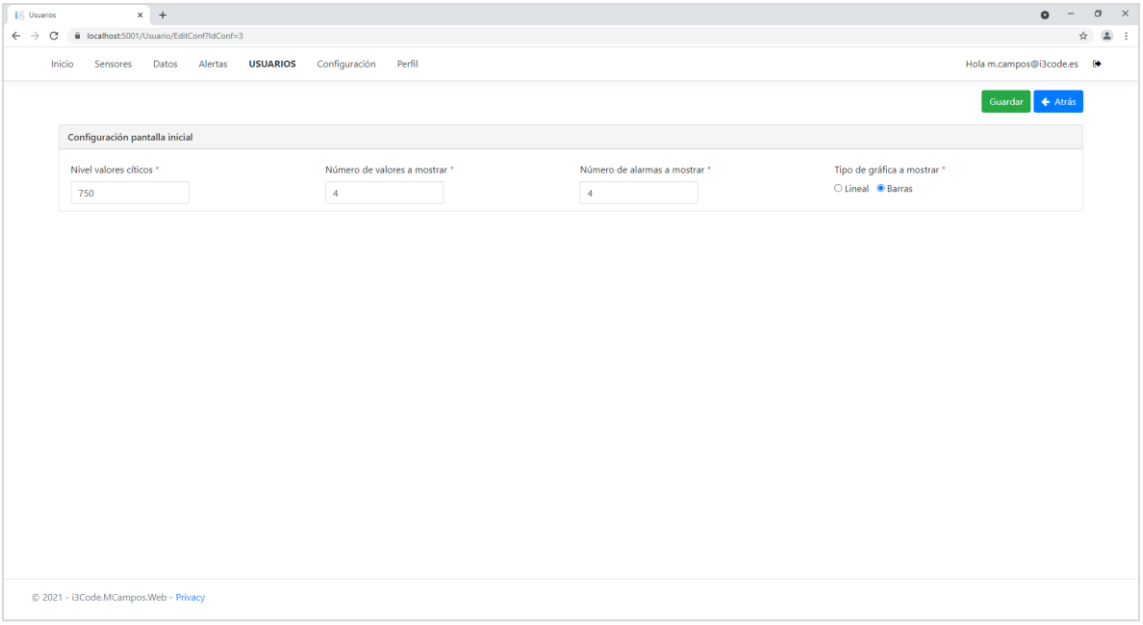

Ilustración 97: Captura de la pantalla de edición de la configuración de usuario (solo administrador)

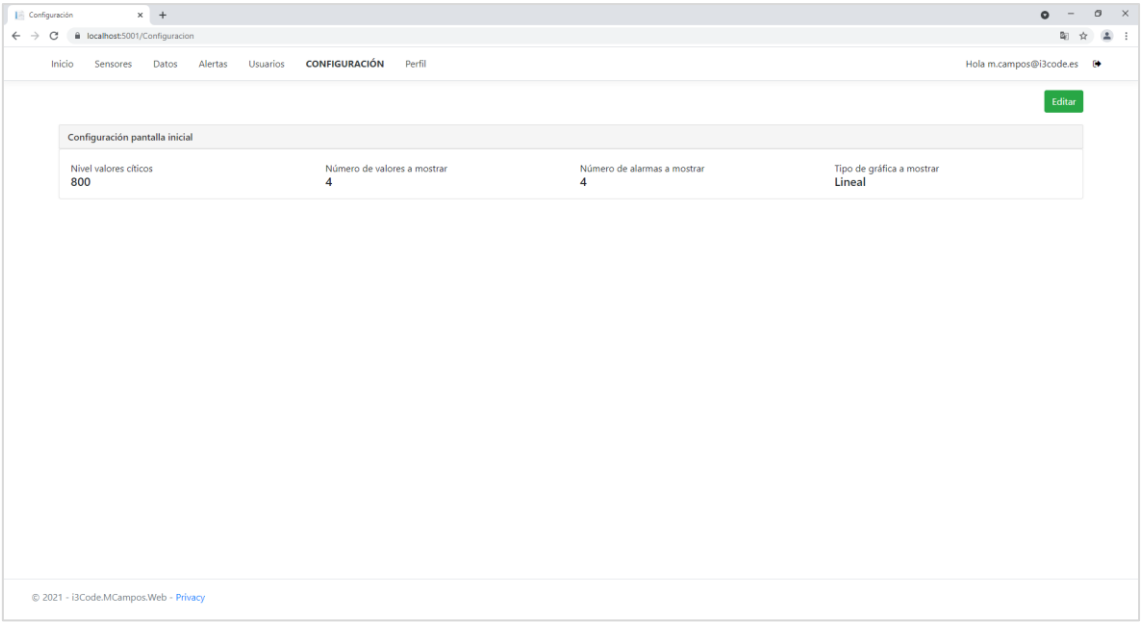

Ilustración 98: Captura de la pantalla de configuración

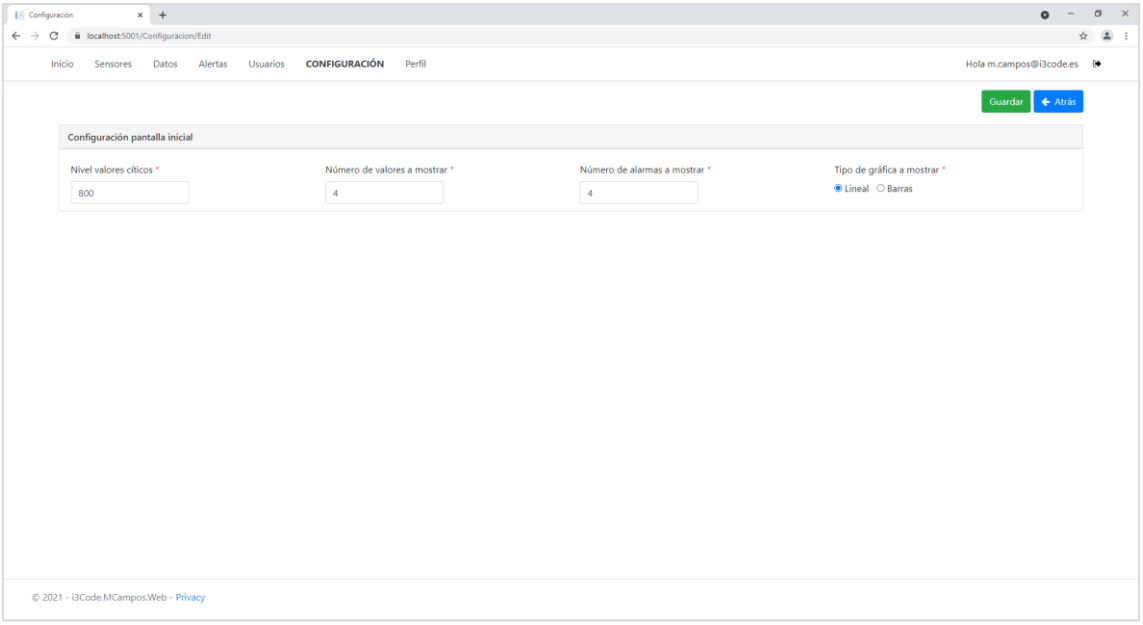

Ilustración 99: Captura de la pantalla de edición de configuración

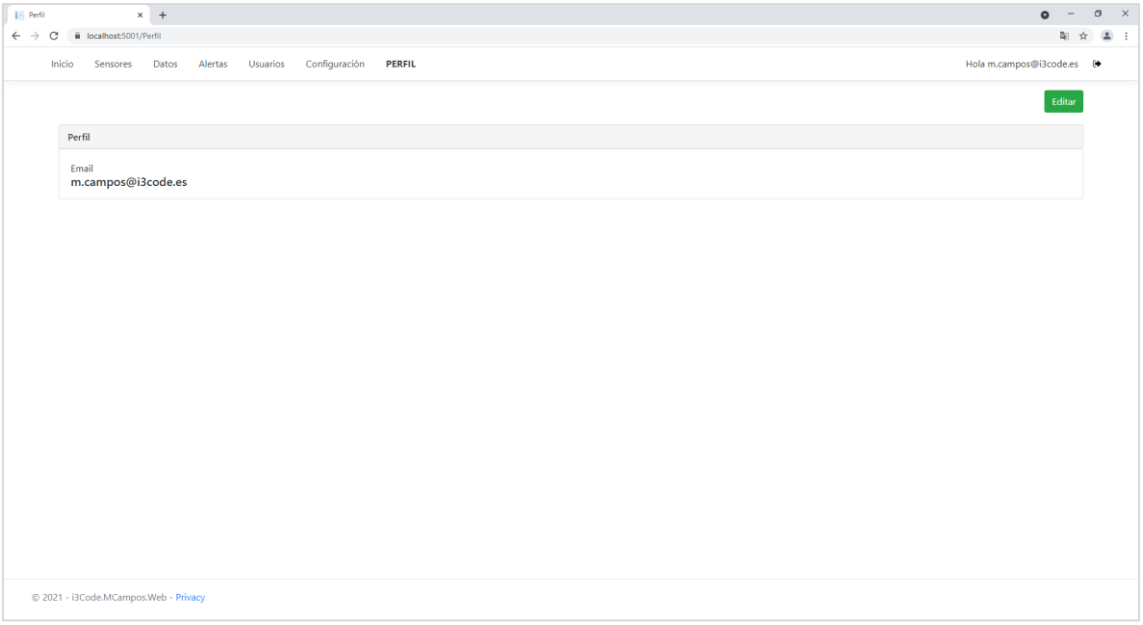

Ilustración 100: Captura de la pantalla de perfil

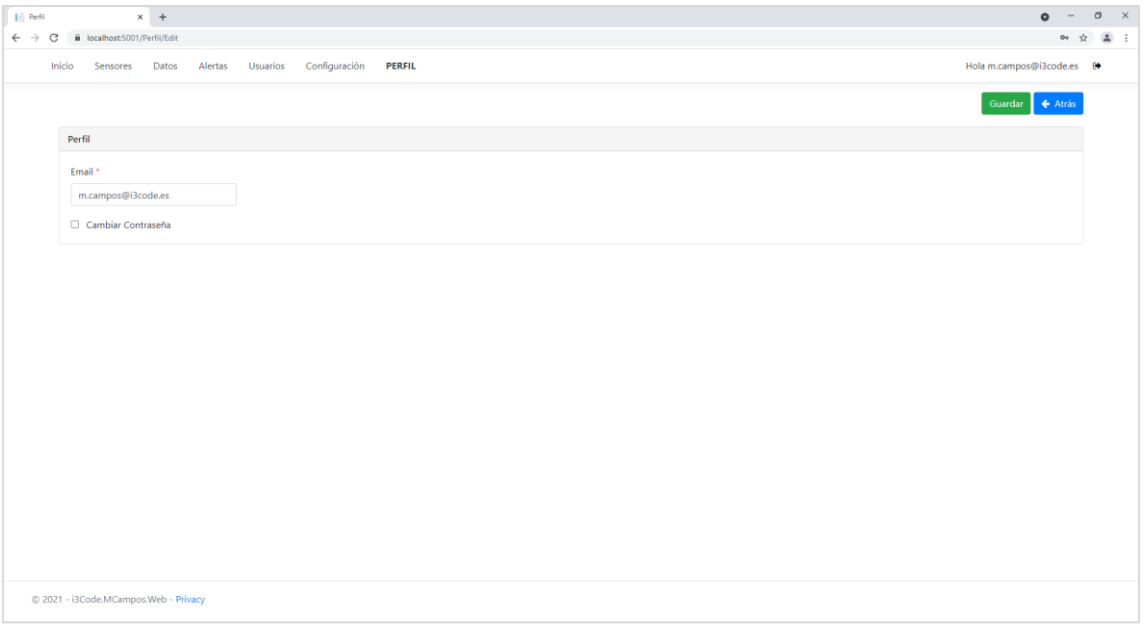

Ilustración 101: Captura de la pantalla de edición de perfil son contraseña

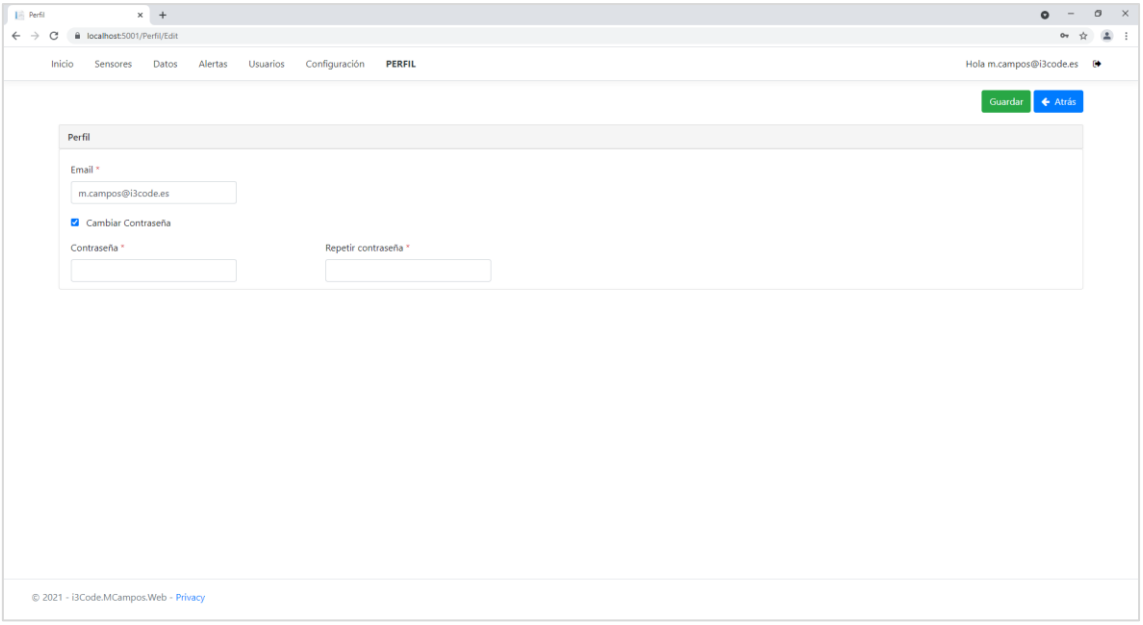

Ilustración 102: Captura de la pantalla de edición de perfil con contraseña

Anexo 13 - Montaje del módulo wifi ESP-12

El módulo wifi ESP-12 dispone de un único modo en el que se puede programar y puede ejecutarse, además no necesita de ningún adaptador para conectarse al ordenador y ser programado o para alimentarse ya que dispone de clavija para cable de tipo micro USB. Para el programa que utiliza los tres leds se ha utilizado el siguiente material hardware:

- Módulo wifi ESP-12
- Sensor Co2 MQ-135
- Tres leds (rojo, naranja o amarillo y verde)
- Tres resistencias de 220 Ω

El montaje es bastante simple, el sensor de Co2 y los leds necesitan una toma a tierra o GND, por tanto, se ha utilizado la placa de conexiones para llevar un cable desde el pin GND del módulo wifi hasta los pines del sensor y los leds. El sensor además necesita estar conectado a una red de $5\mathrm{V}$ y el módulo wifi tiene un pin que provee de tal voltaje por lo que se han unido ambos pines. El último pin del sensor que se debe conectar es su pin $A0$ por el que envía una señal analógica, este pin se ha conectado al pin con el mismo nombre del módulo wifi para que recoja los datos enviados por el sensor. Por último, cada uno de los leds necesita recibir la señal que indique si debe encenderse o apagarse, para eso se utilizan los pines digitales, en este caso los pines $D4$, $D3$ y $D2$ del módulo wifi, pero podían haberse utilizado cualquiera del resto de los pines digitales. De esta forma ya tenemos el montaje hecho, el módulo wifi alimenta al sensor Co2 y es capaz de recoger la información que envía y además está conectado con los leds y puede decidir cuándo encenderlos o apagarlos según la información recibida por el sensor.

Modulo wifi ESP-12	Sensor Co2	Led rojo	Led naranja	Led verde
5V	VCC	-	-	-
GND	GND	Pata corta	Pata corta	Pata corta
A0	A0	-	-	-
D4	-	Pata larga	-	-
D3	-	-	Pata larga	-
D2	-	-	-	Pata larga

Tabla 17: Conexión modulo wifi ESP-12

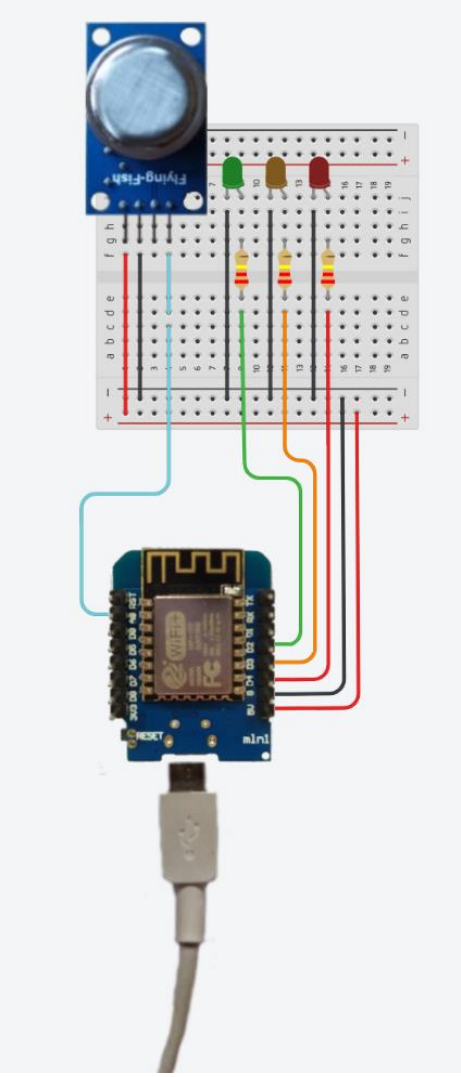

Ilustración 103: montaje módulo wifi ESP-12

Anexo 14 - Programa para calcular el Co2 en ppm y encender leds

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#define sensorPin A0  // Pin del modulo wifi donde se encuentra el sensor
#define ledRojoPin 2 // Pin del modulo wifi donde se encuentra el led rojo
#define ledAmarilloPin 0 // Pin del modulo wifi donde se encuentra el led amarillo
#define ledVerdePin 4 // Pin del modulo wifi donde se encuentra el del verde

//Variables del sensor
float a = 5.5359; // Factor de escala
float b = -0.3481; // Exponente
int Rl = 10500; // Resistencia de carga Rl = entre 10k y 20k(omhio)
float R0 = 284929.49; // Resistencia contante que se desea obtener
float ppm_calculado; // Calculo del PPM
int mediciones = 60; // Mediciones por minuto
int tiempo_espera = 540000; // Tiempo de espera entre mediciones
int adc;  // Lectura de la salida del sensor
int Rs_medio; // Preomedio de las lecturas
float Rs; // Lectura
int valorAlarmaMedio = 500; //Parametro mínimo de alarma, enciende la luz verde
int valorAlarmaAlto = 800; //Parametro mínimo de alarma, enciende la luz verde
int valorAlarmaMuyAlto = 1100; //Parametro mínimo de alarma, enciende la luz verde

// Parámetros configuración WiFi
const char* ssid = "SSID";
const char* password = "password";

// Parámetros configuración broker MQTT
const char* brokerUrl = "13.80.129.109";
int brokerPuerto = 1883;
const char* userMQTT = "monica";
const char* passMQTT = "monica";
const char* topicMQTT = "Co2/Sensor1";
const String clienteId = "Sensor1";

// Instancia al objeto WiFiClient y PubSubClient
WiFiClient clienteESP;
PubSubClient clienteMqtt(clienteESP);

void configuracionWifi() {
  Serial.print("Inicio conexión con la red WiFi ");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);  // Modo estación
  WiFi.begin(ssid, password); // Inicio comunicación
  while (WiFi.status() != WL_CONNECTED) { // Conexión con red WiFi
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Conectado a la red WiFi ");
  Serial.println(ssid);
  Serial.print("Con la IP ");
  Serial.println(WiFi.localIP());
}

void obtenerEnviarCo2() {
  Serial.println("Calculando...");
  int suma = 0;
  for (int j = 0; j < mediciones; j++){ // Recoger valor Rs cada segundo durante 1 minuto
    adc = analogRead(sensorPin);
    Rs = 1024*(Rl/adc)-Rl;
    suma = suma + Rs;
    delay(1000);
  }
  Rs_medio = (suma / mediciones);
  ppm_calculado = pow( ((Rs_medio/R0)/a), (1/b) );
```

```
  Serial.print("El valor del Co2 en ppm es: ");
  Serial.println(ppm_calculado);
  char payload[7]; // Número de caracteres máximo 6 (XXX.XX)
  snprintf(payload, 7, "%d", int(ppm_calculado)); // Convertir un float en array de char
  conectarMQTT();
  clienteMqtt.publish(topicMQTT, payload);  // Enviar mensaje con el valor Co2 en
  Serial.print("Enviado mensaje con Co2 (ppm) ");
  Serial.print(payload);
  Serial.print(" al topic ");
  Serial.println(topicMQTT);
  alarma();
}

void alarma(){
  if(ppm_calculado >= valorAlarmaMedio){
    if(ppm_calculado < valorAlarmaAlto){
      digitalWrite(ledVerdePin, HIGH);
      delay(tiempo_espera);
      digitalWrite(ledVerdePin, LOW);
    }else if(ppm_calculado < valorAlarmaMuyAlto){
      digitalWrite(ledAmarilloPin, HIGH);
      delay(tiempo_espera);
      digitalWrite(ledAmarilloPin, LOW);
    }else{
      digitalWrite(ledRojoPin, HIGH);
      delay(tiempo_espera);
      digitalWrite(ledRojoPin, LOW);
    }
  }
}

void reconnect() {
  while (!clienteMqtt.connected()) {    // Repetir hasta que se conecte
    Serial.print("Intentando conectarse al broker MQTT...");
    if (clienteMqtt.connect(clienteId.c_str(), userMQTT, passMQTT)) {
      Serial.println(" Conectado");
    } else {
      Serial.print("Fallo al conectar al broker, rc=");
      Serial.print(clienteMqtt.state());
      Serial.println(" intentando conectar en 5 segundos");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(9600);
  configuracionWifi();  // Configuración WiFi
  clienteMqtt.setServer(brokerUrl, brokerPuerto); // Configuración MQTT
  pinMode(ledRojoPin, OUTPUT);
  pinMode(ledAmarilloPin, OUTPUT);
  pinMode(ledVerdePin, OUTPUT);
}

void loop() {
  obtenerEnviarCo2();
}

void conectarMQTT(){
  // Gestión conexión MQTT
  if (!clienteMqtt.connected()) {
    reconnect();
  }
  clienteMqtt.loop();
}
```

Anexo 15 - Cantidad de Co2 recomendable en el aire

Para conocer cómo es la calidad del aire según el nivel de Co2, se ha estudiado que el Co2 se mide en ppm (partes por millón). Esta medida se utiliza para la concentración. La fórmula de ésta viene dada por la división de la cantidad de soluto entre la cantidad de la solución. En el caso actual, el soluto sería el Co2 y la solución el aire de la habitación que se está estudiando, por tanto, la formula quedaría algo como:

$$\frac{Cantidad\ de\ Co2}{Cantidad\ de\ aire\ en\ la\ habitación} = ppm\ de\ Co2$$

Una vez conocida la unidad de mediad del Co2, se han investigado los valores de Co2 en ppm que hacen una calidad de aire buena o mala. Se ha observado que varias fuentes coincidían en que en exterior la media de ppm de Co2 es de 400 ppm aproximadamente. En interior a un valor por debajo de los 500 ppm se considera una buena calidad de aire, un valor entre 500 y 800 ppm se considera una calidad media, pero valores que superan los 800 ppm son indicador de falta de ventilación y mala calidad del aire por lo que se recomienza ventilar de forma urgente. En el caso de obtener valores por encima de los 1.000 o 1.200 ppm se considera obligatorio ventilar al máximo y lo más rápido posible ya que estos valores indican falta de ventilación completa. Por tanto, con todos estos datos, se podría construir una tabla de medidas de Co2 en ppm como la siguiente:

Nivel de Co2 (ppm)	Calidad del aire	Nivel de alerta
Hasta 500 ppm	Buena	Bajo
Hasta 800ppm	Media	Medio
Hasta 1100 ppm	Mala	Alto
Más de 1100 ppm	Muy mala	Muy alto

Tabla 18: Calidad del aire según Co2 (ppm)

En la aplicación se ha añadido esta misma tabla como información en el apartado de alertas, para que los usuarios conozcan estas medidas: